Universidad Carlos III de Madrid

Escuela Politécnica Superior

DESARROLLO DE UN MARCO DE TRABAJO PARA JAVA/J2EE: PLATAFORMA PUZZLE

PROYECTO FIN DE CARRERA

INGENIERÍA SUPERIOR INFORMÁTICA

Autor: Vicente Ramos Fernández
Tutor académico: Jesús Carretero Pérez, Universidad Carlos III de Madrid

2009

Para todas las personas que me han apoyado y ayudado durante todos estos años hasta llegar a presentar este proyecto de fin de carrera

Gracias a todos.

Índice

Índice de ilustraciones

Índice de tablas

1 Introducción

En la actualidad existen muchas opciones de diseño y desarrollo de aplicaciones Web/J2EE así como los servicios o facilidades subyacentes a dichas aplicaciones, tales como el acceso a Bases de Datos, Seguridad y facilidades de presentación entre otros y, lo que hace difícil es decidir qué usar y cómo usarlo en los diferentes proyectos.

En el caso particular de Telefónica, y en especial Telefónica I+D, ha pretendido siempre desarrollar los mejores sistemas posibles, entre otros, apostando en el campo de los servicios web que el grupo ofrece; uno de estos proyectos se llama **Plataforma Puzzle.**

En este proyecto, iniciado en el seno de Telefónica I+D y en continuo desarrollo, crecimiento y mejora, actualmente se apoyan ya muchos de los servicios y aplicativos web que ofrece el grupo Telefónica. Estos aplicativos web ofrecen la gestión de las más diversas actividades asociadas a los servicios que el grupo ofrece, tales como contratación, gestión de reclamaciones, facturación y consulta entre otros, para un amplio espectro de clientes de Telefónica, desde grandes empresas hasta el gran público, todos ellos accesibles desde intranets, extranets e internet.

El propósito del Proyecto Final de Carrera se centra en ilustrar una de las partes más fundamentales del citado proyecto, además de ser el responsable de la misma, el denominado **Framework Puzzle como núcleo de la plataforma**, describiendo el proceso de desarrollo en todas sus fases, desde el análisis y diseño hasta la implementación del Framework, el cual ha sido y será destinado como plataforma al desarrollo de aplicaciones web, enfocadas en Java y la plataforma J2EE, con el objetivo de simplificar la complejidad inherente de los desarrollos de Software en estas tecnologías.

1.1 ¿Qué es un Framework?

Conocido el objeto del proyecto y del documento en curso, y teniendo en cuenta que la piedra angular del proyecto parte del concepto **Framework**, vamos a describir, apoyándonos en la conocida **Wikipedia [26],** el citado concepto. Según la Wikipedia, la definición de framework es como sigue:

Un framework, en el desarrollo de software, es una estructura de soporte definida mediante la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto. Representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

Fuera de las aplicaciones en la informática, puede ser considerado como el conjunto de procesos y tecnologías usados para resolver un problema complejo. Es el esqueleto sobre el cual varios objetos son integrados para una solución dada.

Ahora, ¿cuál es el objetivo? Los frameworks son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Después de todo, un Framework no debe ser consciente de todos estos requerimientos sin tener que ser intrusivo con las aplicaciones que permite dentro de sí mismo. A esto, le sumamos la capacidad de extenderse sin prejuicios para diversificar la expresión del programa mismo.

Arquitectónicamente hablando, dentro del ámbito del desarrollo web, no encontramos en que la inmensa mayoría se basa en el **modelo MVC** (Controlador => Modelo => Vista) donde fragmenta o separa la programación del sistema en estas tres capas, la cuales se explican con más detalle más adelante.

En general, y como veremos a lo largo del documento, se irán incorporando conceptos y detalles para un mayor entendimiento del concepto de framework en el desarrollo actual de aplicaciones Web/J2EE basadas en Java, y en particular, la visión y el soporte que proporciona el framework desarrollado y descrito en este documento: ***Puzzle Framework***.

1.2 Resultado

El Framework Puzzle y sus facilidades proporcionan una Aplicación Base para el desarrollo de aplicaciones Web en J2EE. Se trata por tanto de un esqueleto de aplicación que incorpora por defecto funcionalidades básicas y el comportamiento estándar que deben seguir las aplicaciones J2EE elaboradas según los estándares de Telefónica I+D y en general, los estándares de desarrollo de aplicaciones Web en J2EE que tienen como referencia el paradigma MVC.

El objetivo que se pretende alcanzar con este desarrollo es que las aplicaciones web se puedan construir a partir de esta aplicación base sin más que ampliar la funcionalidad específica. De esta manera, todas las aplicaciones se construirán sobre la misma arquitectura, lo que garantizará, entre otras muchas ventajas, que ciertos comportamientos sean homogéneos para todas las aplicaciones.

Este proyecto posee un conjunto de características que son la aplicación de los estándares de desarrollo Web corporativo en Telefónica I+D, así como un amplio número de recursos y herramientas que permiten mejorar la productividad a la hora de afrontar un desarrollo Web a medida y de calidad.

Las aplicaciones generadas a partir de este esqueleto base están asociadas con un completo sistema de seguridad que autorizan la interacción únicamente a los usuarios especificados y a las acciones previamente descritas. Además, asegura que cumplen con los estándares corporativos de desarrollo Web y se beneficiarán de disponer de:

- Una gestión eficiente de los ficheros de configuración.
- Un Escritura en los ficheros de log.
- Una Organización del proyecto en directorios y ficheros
- Una calidad en la organización y estilos de la capa de presentación.
- Una arquitectura interna escalable y fácilmente de mantener.
- Otros

1.3 Estructura y contenido del documento

Este punto pretende ofrecer una breve visión global de los contenidos ofrecidos en este documento. Se explica, de forma resumida, cada uno de los grandes puntos que conforman el escrito.

- Estado del arte: en este punto se proporciona un vistazo general, con carácter introductorio, empezando con una introducción a los marcos de trabajo en general y continuando con la descripción del desarrollo de un Framework y la calidad, todo ello bajo la perspectiva del desarrollo de aplicaciones J2EE.

- Una vez situados en el contexto y objeto del documento, se pasa al detalle del desarrollo del Framework Puzzle como marco de trabajo para el desarrollo de aplicaciones Web/J2EE, desde el qué es y los objetivos que espera cubrir, pasando por su análisis y diseño, sus herramientas y las soluciones para el desarrollo que proporciona, todo ello, siguiendo una metodología básica para el desarrollo software. Estas son:

    - Fase de Análisis del sistema: en este punto se describe la tarea del análisis del sistema, en el sentido más amplio de la palabra, que servirá como base y justificación para el posterior diseño del sistema. En este punto se definen:
        - Análisis del dominio
        - Arquitectura del Sistema
        - Requisitos Software

    - Fase de Diseño del sistema: en este apartado se describen detalladamente cada uno de los componentes que se implementarán en el desarrollo del Sistema mediante diagramas UML, tales como los diagramas de interacción y diagramas de clase.

- Manual de usuario del Framework Puzzle: este apartado se centra en las herramientas y las soluciones para el desarrollo que ofrece así como un pequeño manual de usuario donde se muestra el cómo utilizar Puzzle, que sirve a arquitectos y desarrolladores, como base para la implementación de buenas prácticas en el desarrollo de aplicaciones web.

- Planificación y Presupuesto: se muestra mediante un diagrama de Gannt la planificación que se ha llevado a cabo en el proyecto. Además, se describen los costes iniciales de mantenimiento correctivo/evolutivo.

- Futuras líneas de trabajo: se comentan los puntos en los que se está trabajando y que se desarrollarán en un futuro para mejorar la plataforma. En particular, se detalla el estado actual de la plataforma de desarrollo Puzzle y sugiere las bases para realizar un estudio DAFO (debilidades, amenazas, fortalezas y oportunidades) comparándolo con herramientas o frameworks, principalmente opensource, presentes en el mercado todo ello recogido en el análisis de viabilidad de Puzzle tal que:

    - Análisis de viabilidad de Puzzle:
        - Aplicaciones actuales basadas en la Plataforma Puzzle definidas dentro de marco empresarial Telefónica I+D
        - Análisis del mercado interno y externo
        - Comparativa con productos existentes

- Conclusiones: en este último apartado se evalúa el proyecto desarrollado y se tratan los puntos que hayamos destacado durante el transcurso de todo el proyecto.

2 Definiciones, acrónimos y abreviaturas

Framework	Marco de Trabajo que ofrece una serie de funcionalidades, previamente diseñadas, que nos ayudan en el desarrollo convencional de software.
DAFO	Estudio de las debilidades, amenazas, fortalezas y oportunidades
MVC	Modelo Vista Controlador, patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos.
Software Factories	Factorías de Software
UML	Unified Modeling Language (Lenguaje de Modelado Unificado)
J2EE	Java 2 Enterprise Edition
J2SE	Java 2 Standar Edition
Java	Lenguaje de Programación Orientado a Objeto
JSP	Java Server Pages
Servlet	Programas escritos en Java que se ejecutan en el contexto de un navegador web, en particular, corren dentro del contexto de un contenedor de servlets (servidor web o aplicaciones)
TagLibraries	Librerías de etiquetas que proporcionan y extienden las etiquetas de HTML otorgándole contenido dinámico.
XML	eXtended Markup Language
DTD	Document Type Declaration
HTML	HiperText Markup Language
API	Application Programming Interface
DAO	Data Access Object
WAR	Web ARchives corresponde al empaquetado de los archivos de una aplicación Web
EAR	Enterprise ARchives corresponde al empaquetado de los archivos de una aplicación Web Enterprise
CORBA	Common Object Request Broker Architecture

JTA	Java Transaction API
JOTM	Java Open Transaction Manager
LDAP	Lightweight Directory Access Protocol
JNDI	Java Naming and Directory Interface
TLD	Tag Library Descriptor

Tabla 1 Definiciones, Acrónimos y abreviaturas

3 Estado del arte

3.1 El desarrollo de un Framework y la calidad

Cada vez que comenzamos un nuevo desarrollo, generalmente, podemos identificar una serie de tareas que son habituales y repetitivas. Bien es cierto que, para algunas de estas tareas, son de utilidad las denominadas *Software Factories*, que no son más que un kit que sirve a arquitectos y desarrolladores como base para la implementación de buenas prácticas en el desarrollo de aplicaciones. Pero se quedan ahí, en la base.

El desarrollo de software moderno requiere algo más que una base, requiere una estructura robusta, firme y estable, que no sólo nos de la guía en inicio del desarrollo, sino que nos acompañe, nos obligue y oriente durante todo el proceso de producción, siguiendo una metodología de base y unas buenas prácticas, estos son los denominados **Frameworks**.

Los frameworks son la piedra angular de la moderna ingeniería del software. El desarrollo del framework está ganando rápidamente la aceptación debido a su capacidad para promover la reutilización del código del diseño y el código fuente. Los frameworks son los Generadores de Aplicación que se relacionan directamente con un dominio específico, es decir, con una familia de problemas relacionados.

Un framework, en el contexto del desarrollo de software, es una estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

En definitiva, representa una arquitectura de software que modela las relaciones generales de las entidades del dominio. Además, provee una estructura y una metodología de trabajo la cual extiende o utiliza las aplicaciones del dominio.

3.1.1 Justificación

Los frameworks son diseñados con el intento de facilitar el desarrollo de software, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de software que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional. Por ejemplo, un equipo que usa *Apache Struts* [9] para desarrollar un sitio web de un banco, puede enfocarse en cómo los retiros de ahorros van a funcionar en lugar de preocuparse de cómo se controla la navegación entre las páginas. Sin embargo, no hay que olvidar que el uso de frameworks añade código innecesario y que la preponderancia de frameworks competitivos y complementarios significa que el tiempo que se pasaba programando y diseñando ahora se gasta en aprender a usar frameworks.

3.1.2 Entendiendo los Frameworks: JUnit, el framework de prueba.

Ya hemos dado un ejemplo de las ventajas del uso de un framework para el desarrollo software, pero veamos otro con más detalle, por ejemplo, *JUnit* [7]. Es un framework simple, bien diseñado, para realizar pruebas en Java.

La configuración de JUnit se muestra en la siguiente figura con un diagrama de clases en UML:

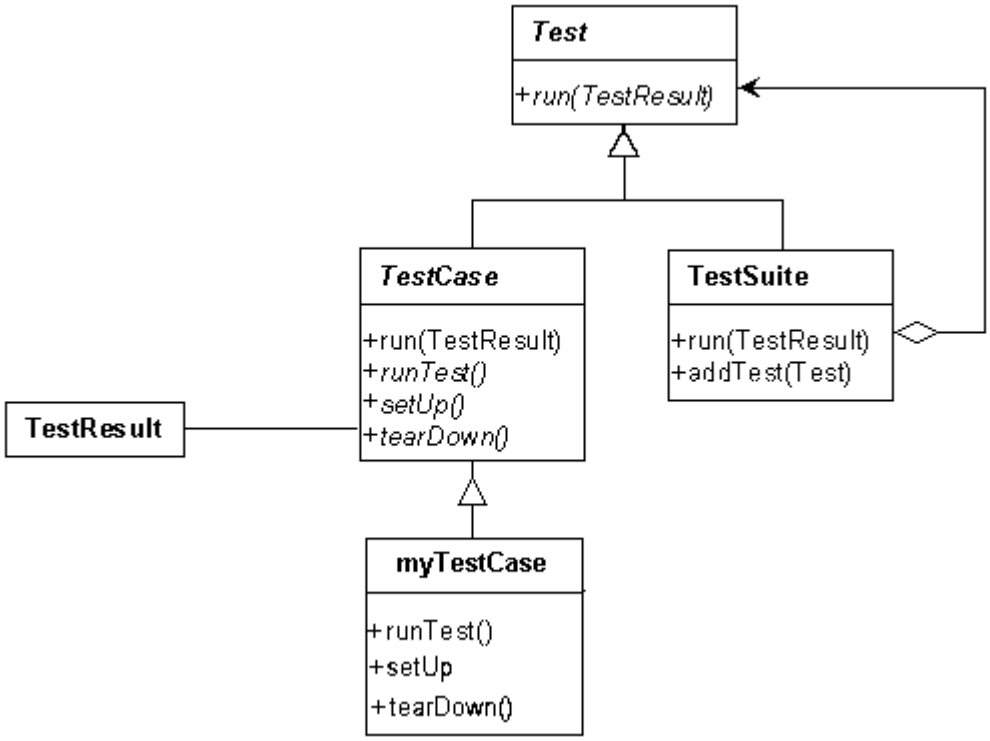

Ilustración 1 Diagrama de clases del framework de JUnit

A partir del framework JUnit, y una vez incorporado a nuestro proyecto, el desarrollador sólo se encargará de incluir las extensiones de las correspondientes clases (myTestCase), donde incluirá su propio código del escenario de prueba correspondiente al negocio, abstrayéndose del flujo de ejecución y de la instanciación de las entidades que intervienen en la prueba, ya que esta actividad es responsabilidad del framework.

Vemos entonces que los puntos calientes o puntos configurables de este framework son: el código de prueba (métodos invocados por runTest), la construcción del escenario de la prueba (setUp), y el código de destrucción (tearDown). Por extensión, las nuevas pruebas que se incorporen sólo se centran en la introducción de la lógica de negocio en particular a probar, el framework se encarga del resto.

3.1.3 Desarrollo del Framework

Las tres etapas principales del desarrollo del framework son:

- Análisis del dominio
- Diseño del framework
- Y la "instanciación" del framework.

El **análisis del dominio** procura descubrir los requisitos del dominio y los posibles requerimientos futuros. Para completar los requerimientos sirven las experiencias previamente publicadas, los sistemas de software similar o existente, las experiencias personales, y los estándares considerados. Durante el análisis del dominio, los puntos calientes o configurables y los puntos congelados o no modificables del framework, por pertenecer al núcleo del mismo, se destapan parcialmente.

La **fase del diseño del framework** define las abstracciones de éste. Se modelan los puntos calientes y los puntos congelados (normalmente y como haremos nosotros, mediante algún lenguaje de modelado como UML [1]), y la extensión y la flexibilidad propuesta en el análisis del dominio se esboza en líneas generales. Según lo mencionado arriba, los modelos del diseño se utilizan en esta fase.

Finalmente, en la **fase de "instanciación",** los puntos calientes del framework son implementados, generando un software del sistema particular. Es importante observar que cada una de estas aplicaciones tendrá los puntos congelados del framework en común, en definitiva, el propio framework.

Las fases del proceso del desarrollo del framework son comparados con las tradicionales fases del diseño orientados al objeto, tal y como se ilustra en la siguiente figura:

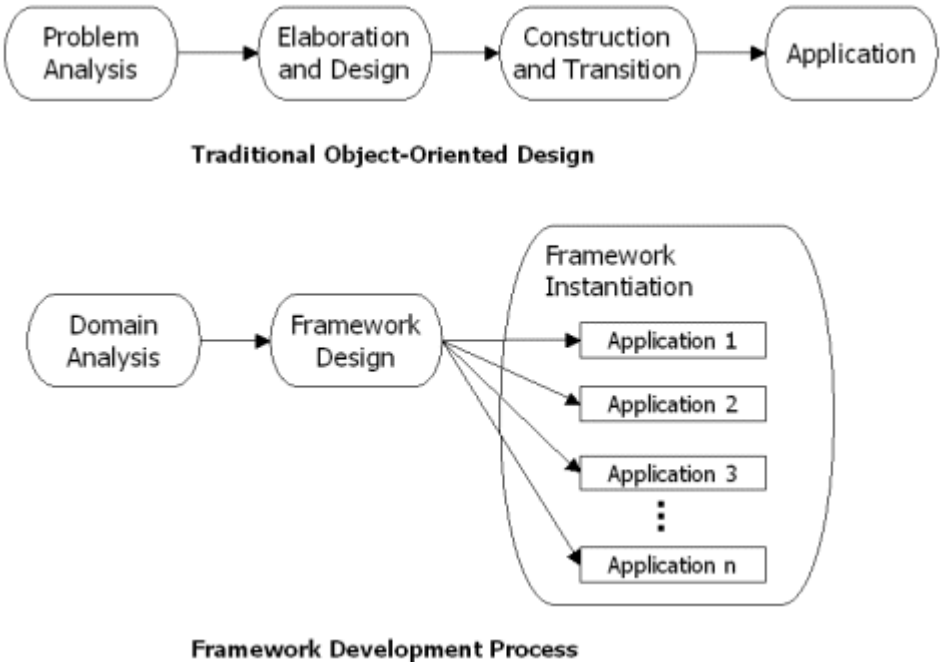

Ilustración 2 Proceso de desarrollo del framework

Podemos observar que existen diferencias entre el desarrollo tradicional Orientado a Objeto frente al desarrollo del framework. En el desarrollo tradicional

Orientado a Objeto, la fase de análisis del problema, también llamada inicio, estudia solamente los requisitos de un solo problema. En cambio, en el desarrollo del framework, captura los requisitos para un dominio entero. Además, el resultado final del desarrollo orientado a objeto tradicional es una aplicación que es completamente ejecutable, mientras que muchas aplicaciones resultan a partir de la fase de "instanciación" del desarrollo del framework. La fase del "instanciación" abarca las fases de construcción y de transición del desarrollo tradicional. Así, la construcción y las fases separadas de la transición están presentes en cada uno de las instancias del framework. Para cada una de las instancias del framework hay un esfuerzo de implementación o puesta en práctica introducido por estas fases.

4 Análisis del sistema

En este apartado se va a realizar el análisis del sistema. En primer lugar, se realiza un análisis del dominio y posteriormente, se justifica y describe el desarrollo del framework, localizando los componentes que conforman el núcleo y los puntos calientes, es decir, la parte configurable instanciable para una aplicación particular basada en Puzzle Framework.

4.1 Análisis del Dominio

Como se ha descrito en el apartado anterior, la primera fase del desarrollo de un framework es la elección del dominio y su análisis. En este sentido, y como se describe en la introducción y objetivos del proyecto, se presenta Puzzle como framework destinado a ofrecer **una plataforma de desarrollo de aplicaciones web, enfocadas en Java y basándonos en la plataforma J2EE,** con el objetivo de simplificar la complejidad inherente de los desarrollos de Software en estas tecnologías.

Actualmente, existen en el mercado infinidad de frameworks destinados al desarrollo de aplicaciones web tales como Java Server Faces (JSF), Spring MVC, Stripes, Struts 2, Tapestry o WebWork entre otros, así como otros innumerables frameworks dependientes, todos ellos generalmente necesarios y usados en este dominio, como la gestión de las trazas (como Log4j) o el acceso a bases de datos (como Hibernate o JDO) resultando difícil la elección de cuál se ajusta mejor a las necesidades de nuestro proyecto.

En este sentido, Telefónica apuesta por la creación de una plataforma, denominada Puzzle, que ofrece un framework para el desarrollo web y que unifica mediante facilidades, usadas según las necesidades del proyecto, cada uno de los frameworks existentes en el mercado, cumpliendo el objetivo fundamental de homogeneizar los aplicativos web existentes en el grupo Telefónica con sus consecuentes ventajas.

Además, otro objetivo de la plataforma es intentan mantener un alto grado de integración con componentes de terceros para cubrir aquellas carencias que la plataforma padezca, de ahí, la continua evolución de la plataforma para cubrir las nuevas necesidades que surgen en los nuevos proyectos.

4.1.1 Contexto

Volviendo al propósito de este proyecto, esto es, diseñar e implementar un Framework para el desarrollo de aplicaciones, enfocadas en Java y la plataforma J2EE, con el objetivo de simplificar la complejidad inherente de los desarrollos de Software en estas tecnologías, analicemos la problemática que se nos presenta a la hora de desarrollar una aplicación web enfocada en las plataformas citadas, o lo que es lo mismo, hagamos el análisis del dominio.

Sabemos por experiencia, lo importante que es la normalización de datos en cualquier aplicación. Los usuarios pueden manejar su información en papel, fichas, en su propia memoria, tenerla duplicada, con incoherencias u omisiones entre otras

muchas, pero una aplicación informática necesita que esa información esté estructurada de un modo conocido para poder manejarla: almacenarla o recuperarla, en general, gestionarla. En este escenario, normalmente lo resolvemos definiendo modelos de datos con una determinada estructura (que habitualmente se convierten en tablas de una base de datos relacional).

Visto esto, veamos ahora el problema en el contexto de la aplicación. ¿Qué ocurre con la información que manejamos los propios analistas y desarrolladores para crear una aplicación? Léase código fuente, librerías, ficheros de configuración, etc., muchas veces parece que la única elección importante es la tecnología concreta a utilizar (como el lenguaje de programación, el servidor de aplicaciones o el gestor de bases de datos) pero, a partir de ahí, surge el problema, donde normalmente cada programador, sin pauta alguna, puede crear su propio maremágnum de ficheros y código fuente.

Entonces, ¿Por qué permitir ese “desorden” en un desarrollo, si estamos tan convencidos de las bondades de estructurar y normalizar la información? Eso es ni más ni menos lo que pretende un framework.

En principio, las premisas y requisitos de los cuales partimos como escenario de aplicación del framework son:

- Aplicaciones Web enfocadas en Java y plataforma J2EE
- Requisitos software característicos de cualquier framework java como compatibilidad con librerías y versiones de JDK.
- Utilización de patrones de diseño y buenas prácticas propuestas por la especificación J2EE, en particular, la aplicación del extendido y consolidado paradigma MVC (Modelo-Vista-Controlador) para el desarrollo de aplicaciones Web.

4.1.1.1 Puzzle y Aplicaciones J2EE

Puzzle está enfocado en Java y la plataforma J2EE. En Java 2 Plataform, Enterprise Edition (J2EE) la cual define el estándar para el desarrollo de aplicaciones empresariales multicapa. La plataforma J2EE simplifica las aplicaciones empresariales aportando unas bases normalizadas, componentes modulares, ofreciendo un conjunto completo de servicios a los componentes, y manejando muchos detalles del comportamiento de las aplicaciones automáticamente, sin programación compleja.

La plataforma J2EE se aprovecha de muchas de las características de la plataforma Java 2, Standard Edition (J2SE), tales como "*Write Once, Run Anywhere*" la portabilidad, la API de JDBC para el acceso a bases de datos, tecnología de CORBA para la interacción con los recursos empresariales existentes, y un modelo de seguridad que protege los datos incluidas las aplicaciones de Internet. Construyendo sobre esta base, J2EE añade un completo soporte para el uso de los componentes Enterprise JavaBeans, Java Servlets API, JavaServer Pages y tecnología XML.

El estándar J2EE incluye las especificaciones completas para garantizar la portabilidad de aplicaciones a través de la amplia gama de sistemas empresariales existentes, capaz de apoyar a la plataforma J2EE. Además, la especificación J2EE ahora garantiza la interoperabilidad de servicios Web a través del apoyo para la WS-I perfil básico.

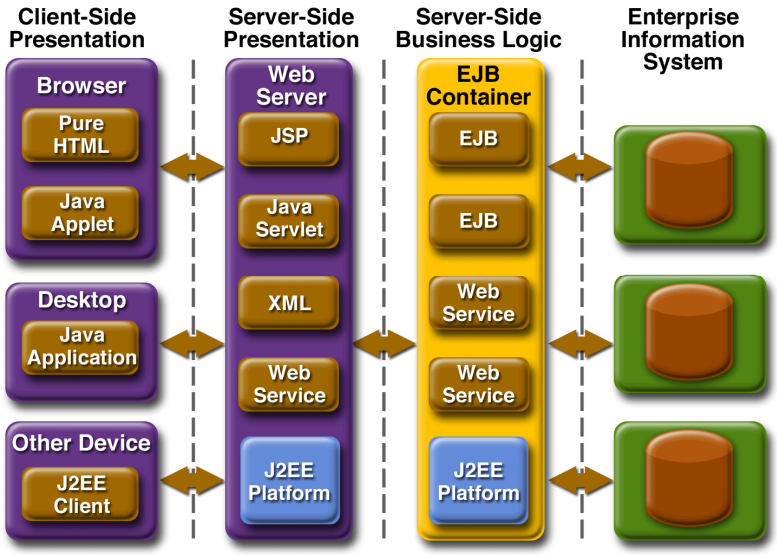

Ilustración 3 Modelo de Aplicación J2EE

Además, no estamos obligados a elegir completamente el conjunto de especificaciones que cubre todo el modelo J2EE, sino sólo las partes que sean útiles para nuestro proyecto o sistema, todo ello gracias a su modularidad, bajo acoplamiento y alta cohesión.

4.1.1.2 Puzzle y MVC

Puzzle es un framework que implementa el patrón de arquitectura MVC en Java, como ya hemos descrito, un framework es la extensión de un lenguaje mediante una o más jerarquías de clases que implementan una funcionalidad y que (opcionalmente) pueden ser extendidas. El framework además, puede también disponer, para facilitar el desarrollo de los componentes vista, TagLibraries (librerías de etiquetas personalizadas).

El esquema MVC (Modelo, Vista, Controlador) es un patrón de arquitectura de software que separa los datos de una aplicación, la interfaz de usuario, y la lógica de control en tres componentes distintos. El patrón MVC se usa frecuentemente en aplicaciones web, donde la vista es la página HTML y el código que provee de datos dinámicos a la página; el modelo es el Sistema de Gestión de Base de Datos y la Lógica de negocio; y el controlador es el responsable de recibir los eventos de entrada desde la vista, encaminarlos a la lógica de negocio apropiada y retornar el resultado asociado a una nueva vista.

- **Modelo**: Esta es la representación específica de la información con la cual el sistema opera, es decir, la lógica del negocio. La lógica de datos asegura la integridad de estos y permite derivar nuevos datos; por ejemplo, no permitiendo comprar un número de unidades negativo, calculando si hoy es el cumpleaños del usuario o los totales, impuestos o importes en un carrito de la compra.
- **Vista**: Este presenta el modelo en un formato adecuado para interactuar, usualmente la interfaz de usuario.

- **Controlador**: Este responde a eventos, usualmente acciones del usuario e invoca cambios en el modelo y probablemente en la vista.

Muchos sistemas informáticos utilizan un Sistema de Gestión de Base de Datos para gestionar los datos. En MVC corresponde o se localiza en el modelo.

Aunque se pueden encontrar diferentes implementaciones de MVC, el flujo que generalmente sigue el control es el siguiente:

1. El usuario interactúa con la interfaz de usuario de alguna forma (por ejemplo, el usuario pulsa un botón, enlace, etc.)

2. El controlador recibe (por parte de los objetos de la interfaz-vista) la notificación de la acción solicitada por el usuario. El controlador gestiona el evento que llega, frecuentemente a través de un gestor de eventos (handler) o callback.

3. El controlador accede al modelo, actualizándolo, posiblemente modificándolo de forma adecuada a la acción solicitada por el usuario (por ejemplo, el controlador actualiza el carro de la compra del usuario). Los controladores complejos están a menudo estructurados usando un patrón de comando que encapsula las acciones y simplifica su extensión.

4. El controlador delega a los objetos de la vista la tarea de desplegar la interfaz de usuario. La vista obtiene sus datos del modelo para generar la interfaz apropiada para el usuario donde se refleja los cambios en el modelo (por ejemplo, produce un listado del contenido del carro de la compra). El modelo no debe tener conocimiento directo sobre la vista. Sin embargo, el patrón de observador puede ser utilizado para proveer cierta indirección entre el modelo y la vista, permitiendo al modelo notificar a los interesados de cualquier cambio. Un objeto vista puede registrarse con el modelo y esperar a los cambios, pero aun así, el modelo en sí mismo sigue sin saber nada de la vista. El controlador no pasa objetos de dominio (el modelo) a la vista aunque puede dar la orden a la vista para que se actualice. En nuestro caso y como muchas otras implementaciones, la vista no tiene acceso directo al modelo, dejando que el controlador envíe los datos del modelo a la vista.

5. La interfaz de usuario espera nuevas interacciones del usuario, comenzando el ciclo nuevamente.

En la siguiente ilustración se muestra el flujo descrito entre las diferentes capas:

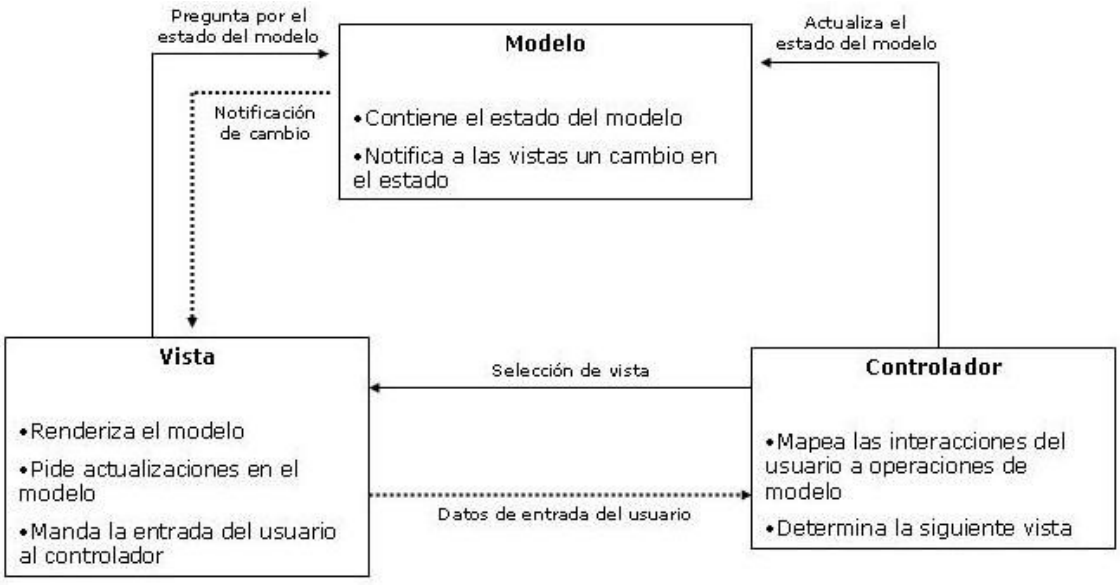

Ilustración 4 MVC (Modelo-Vista-Controlador)

4.1.2 Escenario y componentes

Una vez analizado el contexto, a continuamos con el análisis del dominio, donde vamos a localizar aquellos puntos que corresponden al núcleo, en principio, inmutables aunque con un cierto grado de configuración, y aquellos puntos denominados calientes, es decir, aquellos que serán los implementados por la instanciación particular de una aplicación basada en el framework.

La siguiente ilustración identifica estos elementos:

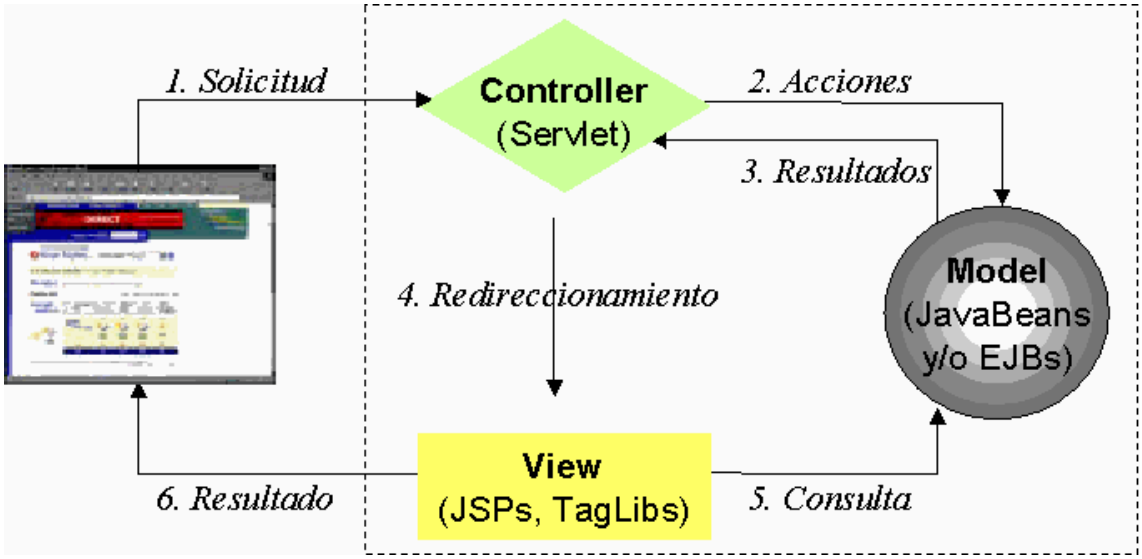

Ilustración 5 Aplicación Web basada en el patrón MVC

A grandes rasgos, el navegador genera una solicitud que es atendida por el *Controller*, el cual se encarga de analizar la solicitud, seguir la configuración que se le ha programado y llamar al *Action* correspondiente pasándole los parámetros enviados. El *Action* instanciará y/o utilizará los objetos de negocio para concretar la tarea. Según el resultado que retorne el *Action*, el *Controller* derivará la generación de interfaz a una o más *JSPs* o vistas, las cuales podrán consultar los objetos del *Model* con el fin de realizar su tarea. Por tanto, detectamos los siguientes componentes:

- Componentes del framework:
    - **Vista**: Opcionalmente puede proporcionar un conjunto de librerías (*TagLibs*) para facilitar la tarea del desarrollo de las páginas web, consiguiendo, entre otras ventajas, la homogeneización de todas las vistas.

    - **Controlador**: Dado un contexto Web, normalmente viene implementado mediante un *Servlet* especializado. Tiene que además ofrecer un fichero de configuración, normalmente en XML, donde se define, en líneas generales, los siguientes requisitos:
        - Flujo de Navegación: Para todas las páginas existe una acción asociada y, dependiendo del resultado de la acción, las diferentes páginas de respuesta asociadas a la respuesta recibida.
        - Configuración de contenedores de datos como Formularios (captura de información) y acciones de negocio.
        - Otras posibles configuraciones basadas en plugins como datasources o servicios de directorio (JNDI).

    - **Modelo**: Componentes abstractos de los cuales extenderán las acciones particulares de la aplicación para que el controlador pueda hacer su trabajo (inicialización, paso de parámetros, invocación de objetos de negocio y resultado)

- Componentes de instanciación: Corresponden únicamente a las particularizaciones del propio negocio de la aplicación web que usa el framework. Estas son:
    - **Vista**: Páginas HTML (normalmente generadas dinámicamente mediante Servlets o JSPs. También se suelen usar TagLibs) u hojas de estilo. Opcionalmente pueden extenderse o particularizar componentes de vista que proporcione el framework.

    - **Controlador**: Únicamente se define el fichero de configuración para definir el flujo de navegación e interacción entre las diferentes vistas y la asociación de correspondencia entre la Vista y el Negocio.

- **Actions y JavaBeans o EJBs**: Componentes que invocan y contienen el negocio particular de la aplicación respectivamente. Estos deben ser extensiones de los componentes abstractos del framework para facilitar la comunicación e integración con el resto de componentes y realizada transparentemente por el framework.

Evidentemente, como todo framework intenta, simplifica notablemente la implementación de una arquitectura según el patrón MVC. El mismo separa muy bien lo que es la gestión del workflow de la aplicación, del modelo de objetos de negocio y de la generación de interfaz.

Logísticamente, separa claramente el desarrollo de interfaz del workflow y lógica de negocio permitiendo desarrollar ambas en paralelo o con personal especializado.

También es evidente que potencia la reutilización, soporte de múltiples interfaces de usuario (Html, sHtml, Wml, Desktop applications, etc.) y de múltiples idiomas, internacionalización, etc.

4.2 Framework Puzzle

4.2.1 ¿Qué es Puzzle?

Puzzle es, básicamente, una plataforma para la construcción eficiente y de calidad de aplicaciones transaccionales de automatización de procesos de negocio y almacenamiento/consulta de información. Podemos definir Puzzle a partir de las siguientes características:

- Puzzle es una plataforma para el desarrollo rápido de aplicaciones Web y J2EE. Para las soluciones J2EE Puzzle recoge su máxima funcionalidad y se denomina como puro J2EE.
- Está centrado en marcos de trabajo o frameworks y componentes de negocio y aplicación.
- Proporciona un conjunto de facilidades y servicios basados en el entorno de ejecución que, como veremos más adelante, define las diferentes soluciones de arquitectura para las aplicaciones web basadas en Puzzle.
- Está basado en patrones de diseño y buenas prácticas J2EE para el desarrollo de aplicaciones

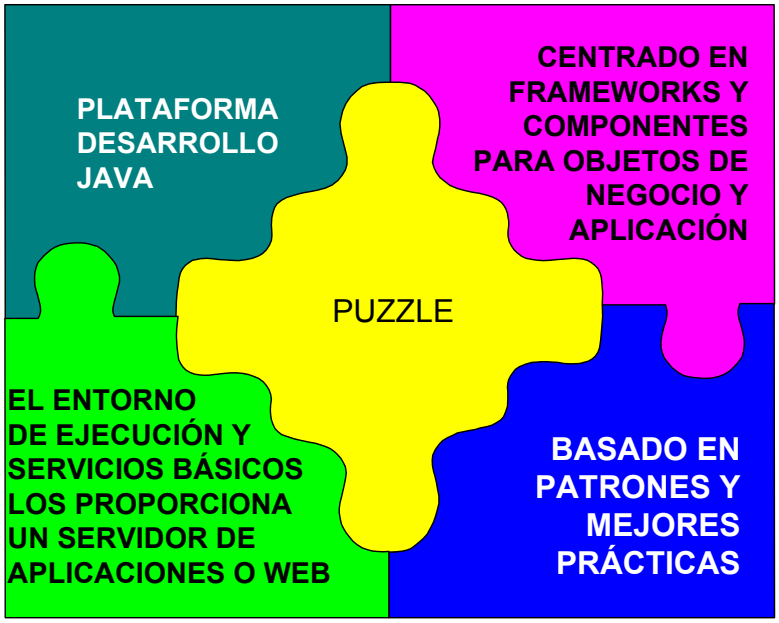

Ilustración 6 Plataforma Puzzle

4.2.2 Historia

La plataforma surgió, inicialmente, en el seno de la gerencia 2730 y se aplicó, inicialmente, a la construcción de la gama de productos HERCULES (Aplicación Web del sector sanitario para la Gestión Clínica de Pacientes). Posteriormente, se ha ido aplicando a otros proyectos, en su totalidad o parcialmente. Igualmente, algunos de sus componentes pasaron a formar parte de la lista de componentes de la plataforma que se denominó, en su momento, miJava.es

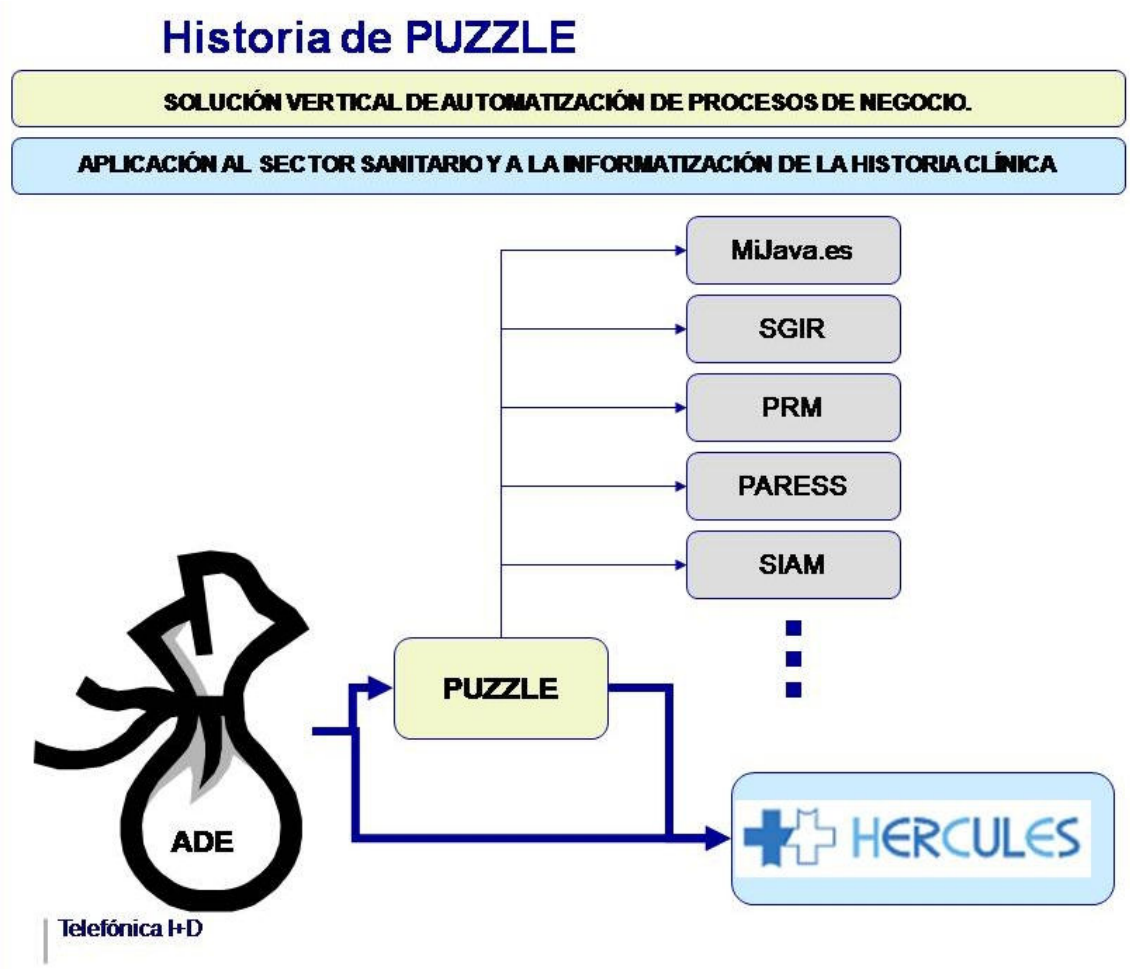

Ilustración 7 Orígenes de Puzzle: Justificación

4.2.3 Objetivos

Los objetivos de Puzzle se enfocan mediante los siguientes puntos de vista:

- De cara al desarrollo
    - Concentración de la problemática específica de cada aplicación, donde el desarrollo de los componentes u objetos de negocio y aplicación son independientes de la infraestructura web como protocolos, dispositivos, despliegue y presentación entre otros.
    - Proporciona un conjunto de herramientas que facilitan las tareas al desarrollador como generadores de código y ficheros de configuración basados en el estándar XML
    - Acorta los tiempos de desarrollo proporcionando una construcción rápida y eficiente
    - Al definir un marco de trabajo fuerza el uso de las reglas y buenas prácticas para el desarrollo de aplicaciones Web basándose en patrones de diseño y el modelo de aplicación MVC (Modelo Vista Controlador)

  - Existe una metodología de desarrollo documentada.
  - Facilita los cambios en las aplicaciones y proporciona una realimentación a partir de la experiencia de producción

- De cara al sistema en producción
  - Escalable, alta disponibilidad, reparto de carga entre otros
  - Fácil mantenimiento, explotación y extensión

- De cara al negocio
  - Reducción de costes
  - Reducción de "*Time to Market*"
  - Fomenta la inversión en soluciones y no en plataformas
    - Planteamiento de soluciones óptimas.
    - Reutilización de soluciones anteriores satisfactorias.
    - Abstraer las soluciones para ampliar su aplicación
  - Competitividad
  - Base para la realización de productos en diversas áreas

4.2.4 Principios de Puzzle

La plataforma Puzzle cumple los estándares J2EE, que constituye el estándar de más amplio apoyo en la industria para la construcción de aplicaciones transaccionales y de misión crítica, huyendo de atajos proporcionados por proveedores y servidores de aplicación, permitiendo así desligar la aplicación construida sobre Puzzle del suministrador. La adhesión al estándar J2EE significa:

- Alineamiento con la industria
- Independencia del Hardware y Software
- Sencilla migración e integración con los diferentes servidores web y de aplicación del mercado
  - Comerciales: BEA WebLogic, Oracle IAS, Sun One, etc.
  - Libre distribución: JBoss, Tomcat, etc.
  - Propios
- Delegación de las funciones a los contenedores (EJB o Servlet)

Además, Puzzle se basa en la aplicación de patrones y paradigmas de diseño más aceptados y utilizados en la industria promoviendo las buenas prácticas en el desarrollo software (patrones como MVC o DAO entre otros). Los patrones usados en Puzzle, aunque deben subyacer al programador Puzzle, son los mostrados en la siguiente figura:

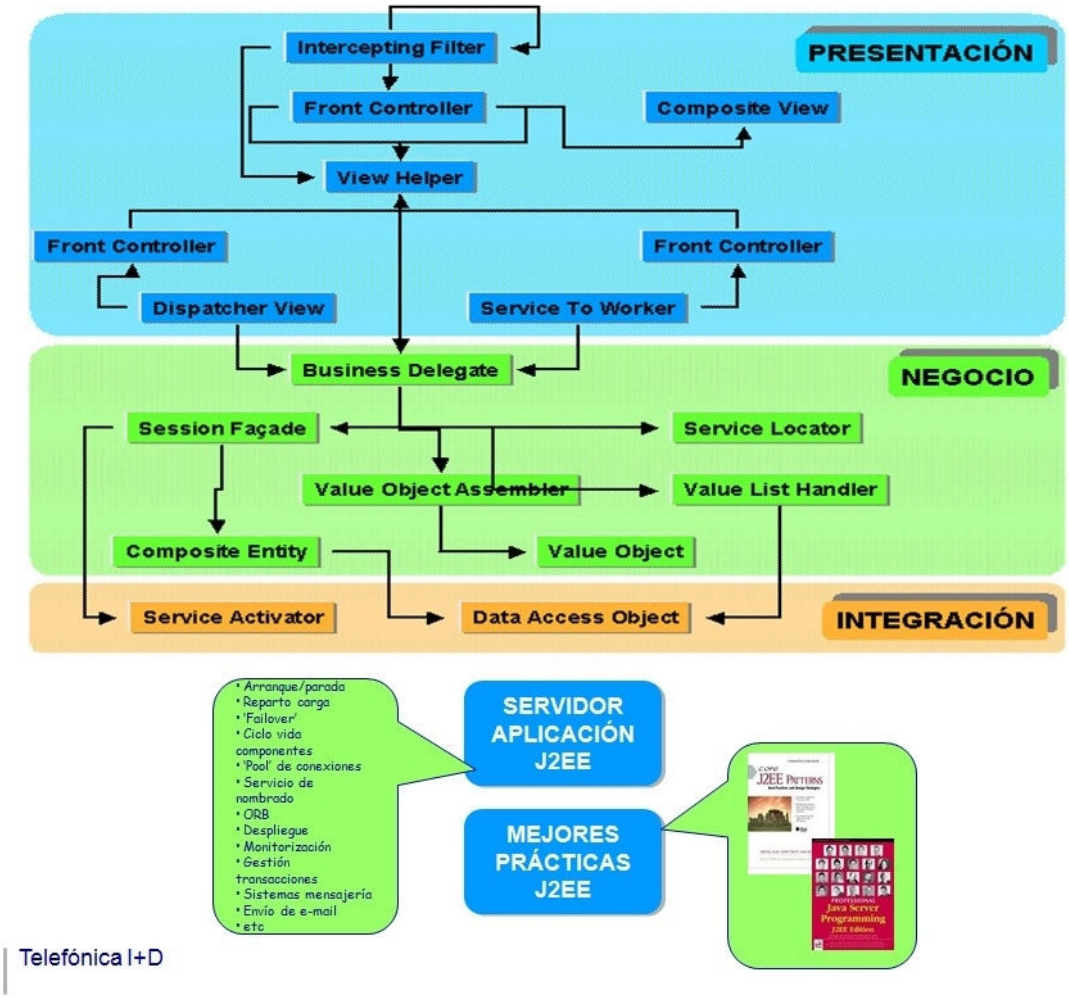

Ilustración 8 Patrones de Diseño y Mejores Prácticas

4.2.5 El planteamiento de Puzzle

Puzzle intenta proporcionar la base para construir aplicaciones flexibles que permiten realizar modificaciones en tiempo de ejecución o con un esfuerzo de desarrollo mínimo. Sin embargo, con frecuencia, los componentes flexibles con capacidad de cambio en tiempo de ejecución tienden a ser más complejos y lentos.

Dado que Puzzle pretende ser la base para aplicaciones de misión crítica se intenta que esa flexibilidad no sea a costa de empeorar las prestaciones. Como un equilibrio adecuado entre prestaciones y flexibilidad, en ocasiones se opta por la generación automatizada de código a partir de ficheros de descripción (el caso más típico de esto se da en el acceso a datos, componente DAO). Un mero cambio en el fichero de descripción y una regeneración de código en tiempo de desarrollo pero con un esfuerzo mínimo.

Las aplicaciones construidas con Puzzle tienen la estructura mostrada en la figura:

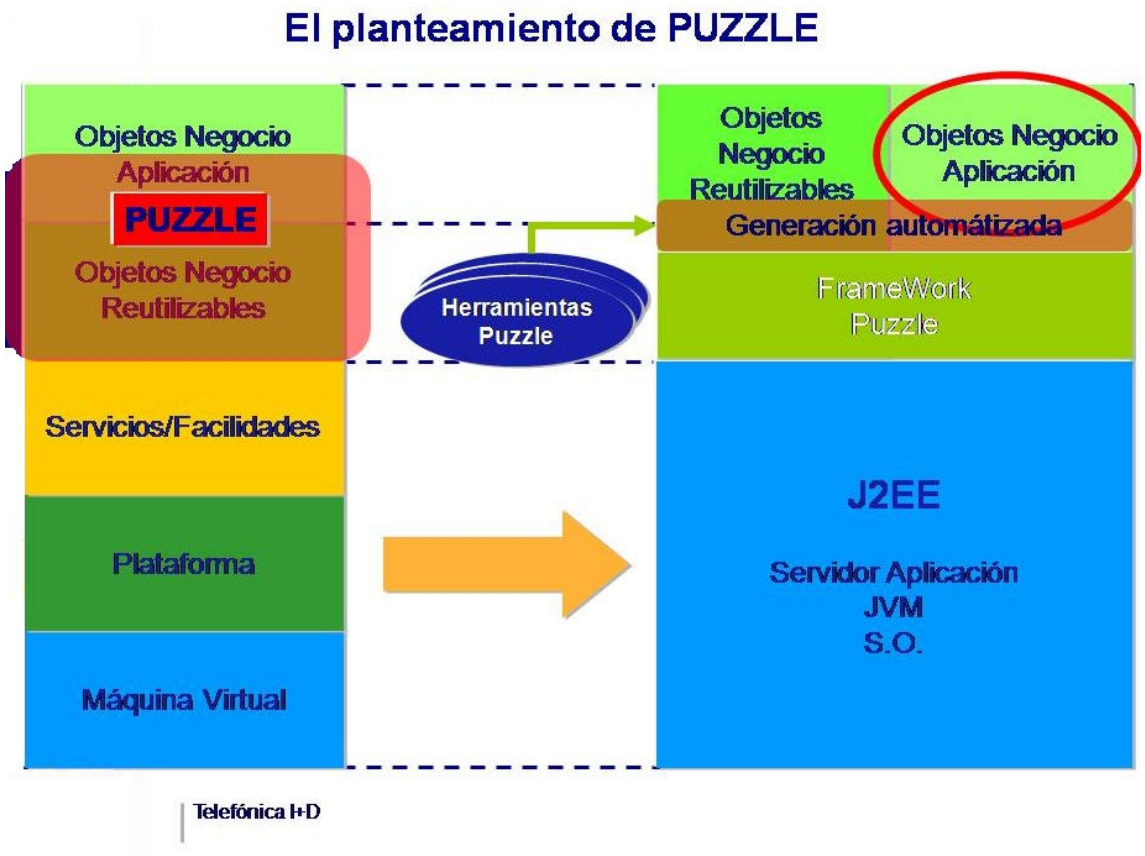

Ilustración 9 Planteamiento de Puzzle

Toda aplicación Puzzle se apoya en contenedores J2EE. La combinación de Java/J2EE proporciona independencia del HW y Sistema Operativo, servicios básicos y facilidades de administración, despliegue y alta disponibilidad.

Sobre esta base, se apoyan los siguientes elementos de Puzzle:

- **Framework Puzzle**: Conjunto de objetos que implementan la mayor parte de los patrones de diseño y proporcionan una infraestructura básica de la aplicación y fuerza el cumplimiento de buenas prácticas de diseño.
- **Componentes generados**: Código generado automáticamente a partir de ficheros descriptores.
- **Herramientas Puzzle**: Conjunto de aplicaciones que se utilizan en tiempo de desarrollo y que se utilizan, fundamentalmente, para la generación automatizada de código.
- **Objetos de negocio reutilizables (servicios y facilidades)**: Componentes reutilizables que proporcionan funcionalidad final.

Además, la construcción de la aplicación implica la construcción de objetos de negocio o componentes específicos de la aplicación.

4.2.6 Framework Puzzle y sus Componentes

La arquitectura básica de Puzzle es la que se muestra en la siguiente figura:

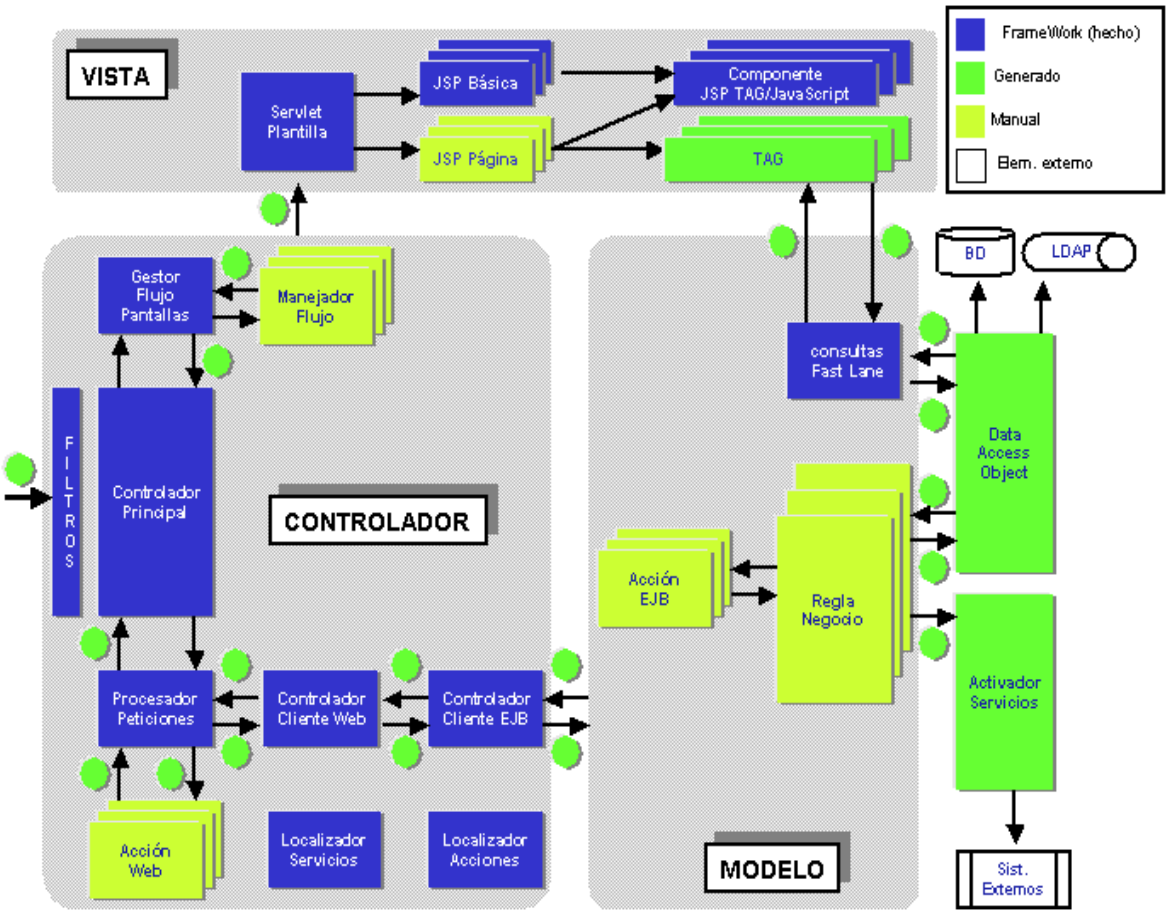

Ilustración 10 Framework Puzzle

El FrameWork Puzzle proporciona una serie de componentes que constituyen el esqueleto que sustenta el funcionamiento de toda aplicación Puzzle.

Entre esos componentes, cada uno encargado de una tarea, circulan una serie de objetos que transportan información. Dichos objetos, representados por círculos en la figura, así como otros componentes adicionales, como los de acceso a datos y activador de servicios, son generados automáticamente mediante ficheros descriptores usando una serie de generadores diseñados específicamente para Puzzle.

4.2.6.1 Facilidades Puzzle

Aunque las facilidades de la plataforma están fuera del ámbito del documento, a continuación se describen las diferentes facilidades y herramientas que proporciona Puzzle como plataforma, además del propio Framework como núcleo de la plataforma.

Facilidad	Descripción	Disp.	Observaciones
Librería javascript	Conjunto de componentes javascript orientados a mejorar las pantallas y formularios HTML		
Generador gráficos	Manejo de gráficos conforme a estándar SVG (estándar W3C). El estado actual es muy básico y no se considera un componente acabado de PUZZLE.		
Generador documentos	Generación de un documento (actualmente PDF) a partir de una descripción XML con toma de datos desde una fuente de datos.		
Gestor transaccional	Gestión de transacciones. Abstrae del sistema transaccional empleado. Actualmente PUZZLE se apoya en la implementación JTA del servidor de aplicación.		
Svc. Configuración (Admin. y monitor.)	Servicio basado en JMX que permite la monitorización de parámetros y su cambio 'en caliente'		
Svc. Firma	Conjunto de librerías y utilidades para la incorporación de la firma digital a los desarrollos.		Actualmente existe una base en fase beta.
Servicio seguridad	Servicio que soporta tareas de autenticación y autorización de forma independiente del servidor de aplicaciones		
Servicio JMS	Abstracción del tratamiento de mensajes basada en JMS pudiendo comunicarse o no con MDBs		Actualmente se apoya en las implementaciones de JMS proporcionados por el servidor de aplicación
Servicio auditoría	Automatización del trazado tanto a efectos de monitorización de la aplicación como de cumplimiento LOPDCP.		La implementación actual es básica y contempla seguimiento de acciones de navegación.
Servicio de log	Completamente soportado por log4j.		
Acceso a Datos (DAO)	Capa de acceso a distintas fuentes de datos (SGBDR, ficheros XML, LDAP, etc) según el patrón DAO. Se incluyen también las herramientas de generación automática de código asociadas.		
Capa movilidad avanzada	Capa que permite la comunicación entre elementos móviles y un servidor, detectando pérdidas de conectividad y sincronizando datos.		Mantiene alguna dependencia de HERCULES Emergencias

Tabla 2 Facilidades Puzzle

Estas facilidades se presentan como componentes totalmente reutilizables.

Facilidad	Descripción	Disp.	Observaciones
Generadores de código	Conjunto de generadores básicos de objetos PUZZLE. Incluye el generador DAO		
Gestión de Tablas Maestras	Generación automatizada de código para gestión de listas de validación, listas de parámetros, etc. Soporta la lógica de negocio y datos (dentro del framework PUZZLE). La presentación es dependiente de la aplicación.		

Tabla 3 Herramientas Puzzle

4.2.7 El proceso de desarrollo con Puzzle

Puzzle proporciona una estructura consistente, y mediante el modelo MVC Arquitectura Modelo - Vista - Controlador), típico en el desarrollo de aplicaciones Web, impone una separación funcional que hace que las aplicaciones sean más fáciles de mantener y extender.

El desarrollo de aplicaciones con el framework Puzzle presenta los siguientes frentes:

- Desarrollo de los elementos de la capa de presentación.
- Desarrollo de los elementos de negocio y de aplicación de la capa de modelo.
- Desarrollo de los elementos de comunicación entre las diferentes capas.
- Definición y modificación de los ficheros de configuración que usa la capa controlador.

Todos ellos están basados en patrones de diseño (Ilustración 8: Patrones de Diseño) y buenas prácticas de desarrollo J2EE.

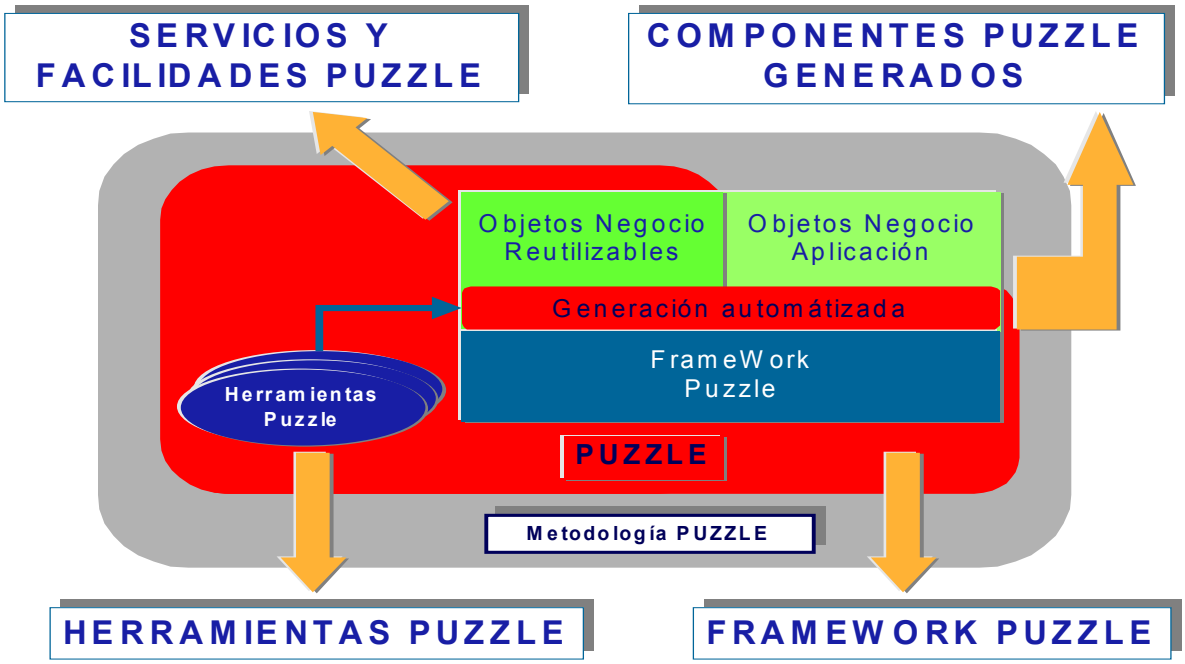

Ilustración 11 Elementos de Puzzle

Un vez conocidos los elementos, a continuación, se ilustran los pasos que tienen que darse para el desarrollo de una aplicación Web basada en Puzzle Framework, desde la captura de requisitos del sistema hasta la entrega del mismo, esto es, el ciclo de vida de un proyecto software basado en Puzzle.

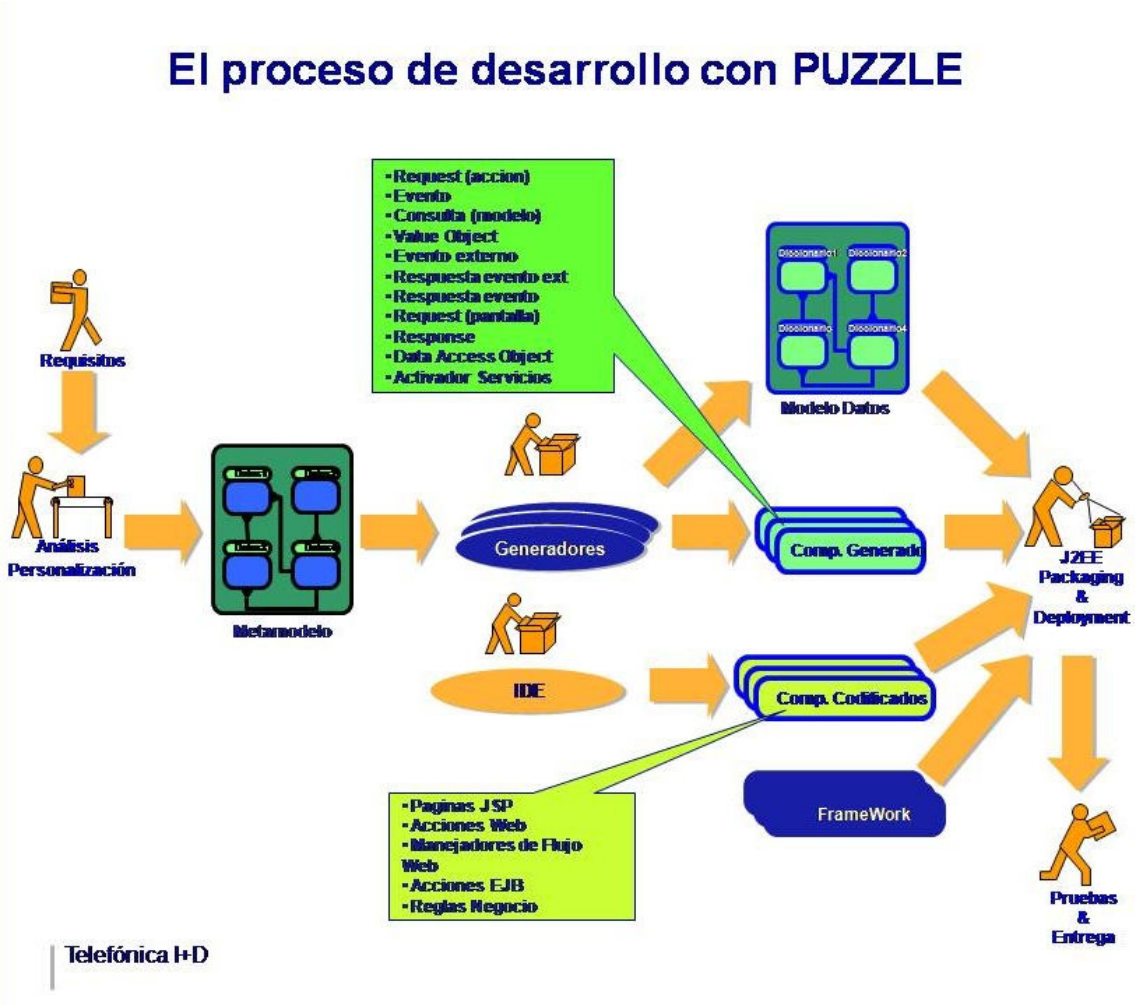

Ilustración 12 Proceso de desarrollo con Puzzle

Podemos observar la importancia del framework Puzzle, que actúa como "pegamento" que conforman las piezas o componentes, así como las herramientas de generación de los componentes particulares de la aplicación y su dominio, aquellas que particularizan los puntos calientes del framework y que contienen la lógica particular del negocio.

4.3 Requisitos Software

En este apartado, como parte del análisis del sistema, se describen los requisitos funcionales, de comprobación, de aceptación de pruebas y de documentación, como requisitos software fundamentales que debe de cubrir el sistema.

4.3.1.1 Requisitos funcionales

Identificador	RS-F001		
Descripción	Los requisitos mínimos hardware se alinean a los requisitos mínimos hardware que requiere la máquina virtual java (definida en documentación del JDK concreto definido por Sun Microsystems).		
Necesidad	Esencial ☒	Deseable ☐	Opcional ☐
Prioridad	Alta ☒	Media ☐	Baja ☐
Estabilidad	Alta ☒	Baja ☐	
Fuente	Cliente		

Identificador	RS-F002		
Descripción	Requiere un JDK 1.2 o superior.		
Necesidad	Esencial ☒	Deseable ☐	Opcional ☐
Prioridad	Alta ☒	Media ☐	Baja ☐
Estabilidad	Alta ☒	Baja ☐	
Fuente	Cliente		

Identificador	RS-F003		
Descripción	Requiere librerías para el desarrollo web que cumplan la especificación Java Server Pages 1.2 o superior y API Servlet 2.3 o superior.		
Necesidad	Esencial ☒	Deseable ☐	Opcional ☐
Prioridad	Alta ☒	Media ☐	Baja ☐
Estabilidad	Alta ☒	Baja ☐	
Fuente	Cliente		

Identificador	RS-F004		
Descripción	Requiere librerías para el procesamiento de los ficheros de configuración XML que cumplan la especificación XML 1.0.		
Necesidad	Esencial ☒	Deseable ☐	Opcional ☐
Prioridad	Alta ☒	Media ☐	Baja ☐
Estabilidad	Alta ☒	Baja ☐	
Fuente	Cliente		

4.3.1.2 Requisitos de comprobación

Identificador	RS-C001		
Descripción	Se comprobará que cumple los requisitos funcionales mediante la compilación y ejecución de un proyecto o aplicación web basada en el framework.		
Necesidad	Esencial ☒	Deseable ☐	Opcional ☐
Prioridad	Alta ☒	Media ☐	Baja ☐
Estabilidad	Alta ☒	Baja ☐	
Fuente	Cliente		

4.3.1.3 Requisitos para la aceptación de las pruebas

Identificador	RS-ACP001		
Descripción	El sistema basado en el framework deberá superar con éxito todas las pruebas realizadas por el equipo de desarrollo durante la implementación del sistema.		
Necesidad	Esencial ☒	Deseable ☐	Opcional ☐
Prioridad	Alta ☒	Media ☐	Baja ☐
Estabilidad	Alta ☒	Baja ☐	
Fuente	Cliente		

4.3.1.4 Requisitos de documentación

Identificador	RS-NF-DOC001		
Descripción	Existirá un manual de ayuda que constará de las siguientes secciones obligatorias: • Manual de programador de ayuda al desarrollo. • Manual de errores catalogados y restricciones de uso • Documentación mediante JavaDoc.		
Necesidad	Esencial ☒	Deseable ☐	Opcional ☐
Prioridad	Alta ☒	Media ☐	Baja ☐
Estabilidad	Alta ☒	Baja ☐	
Fuente	Cliente		

5 Diseño del Sistema

En este apartado, una vez realizado el análisis, se va explicar el diseño del sistema.

Como se ha descrito en análisis del framework, Puzzle sigue el esquema MVC (Modelo, Vista, Controlador) y, en este sentido, el FrameWork Puzzle proporciona una serie de componentes que constituyen el esqueleto que sustenta el funcionamiento de toda aplicación Puzzle.

Desde el punto de vista de las capas a las que da soporte el framework, para el desarrollo de aplicaciones, nos encontramos: capa de presentación, capa controlador, capa de negocio y capa de acceso a datos. Además existe, por otro lado, un conjunto de componentes técnicos o facilidades que dan soporte a la infraestructura por separado.

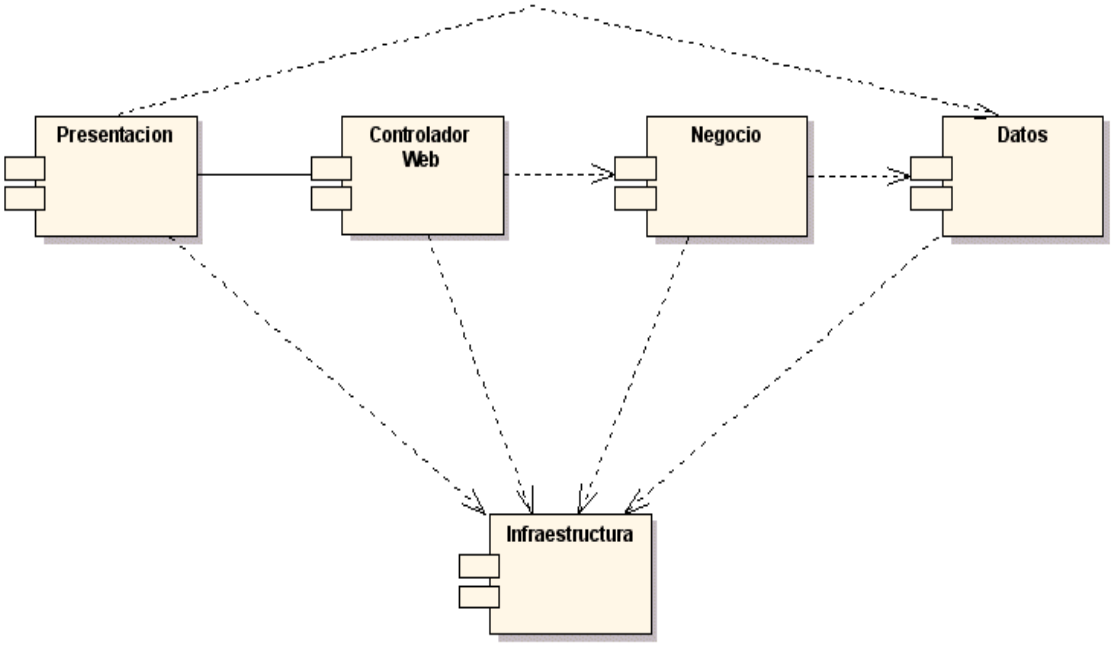

Ilustración 13 Esquema de Componentes general de Puzzle

A continuación, describimos las diferentes capas según Puzzle:

- **Capa de Presentación**: Puzzle incorpora dentro de la capa controlador un elemento (servlet) que se encarga de gestionar la capa de presentación y permite el trabajo con plantillas (templates). De esta manera, Puzzle descarga la programación de las interacciones que en última instancia desencadenan operaciones de negocio de las interacciones que se encargan de cargar datos para la capa de presentación.

  Por otro lado, esta estructura de controlador en la presentación permite que Puzzle pueda trabajar con diferentes tecnologías de presentación, sin más que incluir el controlador de presentación correspondiente.

- **Capa Controlador**: (Módulo Web) Una de las partes que proporciona Puzzle en el desarrollo de aplicaciones J2EE es una infraestructura para la programación con Servlets.

  La infraestructura que se propone se adapta perfectamente a las recomendaciones descritas en los patrones de diseño para el desarrollo de aplicaciones J2EE [25] y se adapta al estándar.

- **Capa de Negocio**: El acceso a la capa de negocio se implementa sobre un pseudo-controlador que interactúa con los objetos de negocio. Podemos observar en la siguiente figura la estructura de acceso a la capa o lógica de negocio.

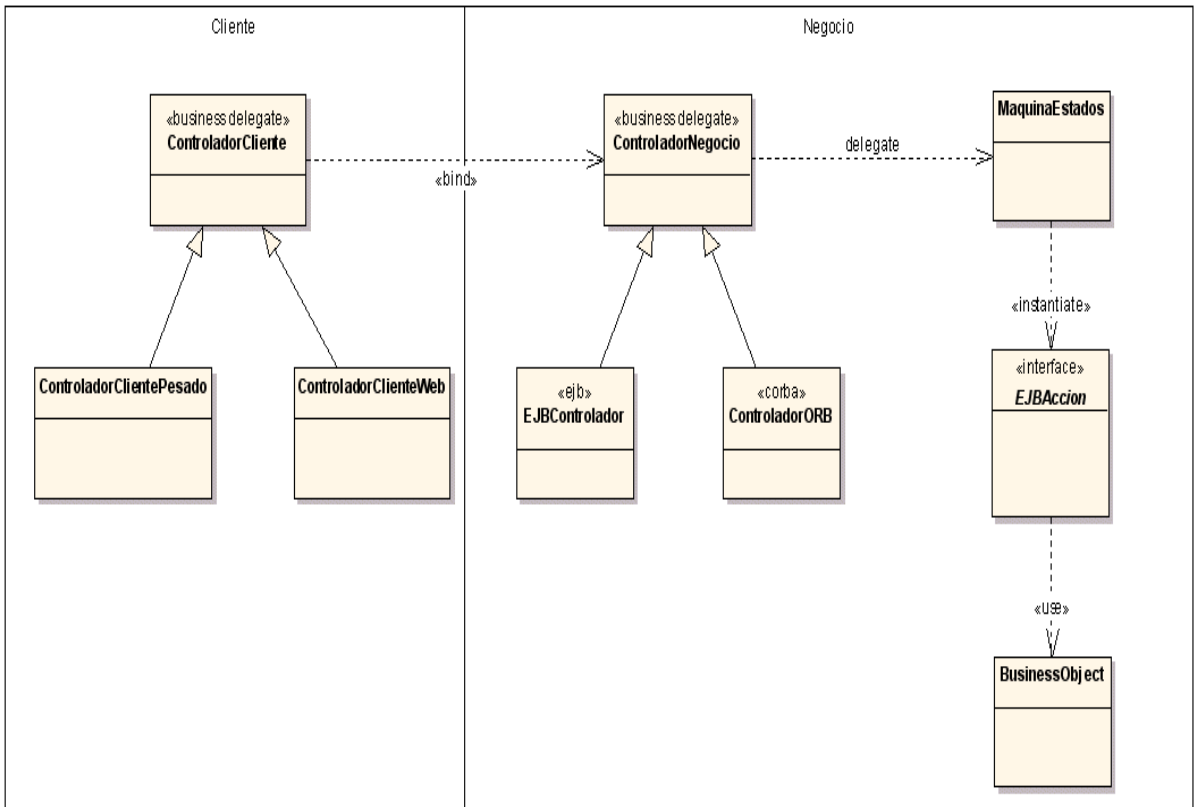

Ilustración 14 Modelo de acceso a la capa de negocio de Puzzle

A través de la infraestructura anterior, Puzzle hace transparente la invocación a la capa de negocio: se instancia un evento en la capa controlador web que automáticamente es encaminado a la capa de negocio y procesado para invocar los objetos de negocio correspondientes. La aplicación de este modelo de propagación de eventos a la capa de negocio en aplicaciones concretas ha demostrado que, en última instancia, la capa de negocio se acopla demasiado con la capa de presentación.

Por otro lado, puesto que la relación entre la clase *ControladorCliente* y la clase *ControladorNegocio* tiene cardinalidad 1 – 1, se limita a que la conexión cliente tenga un único punto de acceso remoto. Esquemas

donde un cliente deba conectarse a varios servicios remotos localizados en diferentes ubicaciones no se podrían implementar con el esquema planteado actualmente en Puzzle, salvo que se haga desde la capa de negocio. En este último caso, Puzzle no define ninguna infraestructura para el acceso a estos servicios remotos.

Por las razones argumentadas en los dos párrafos anteriores, lo que en última instancia se consigue es que la capa de negocio se convierta en una prolongación de la capa de presentación.

En el caso en que una aplicación implementada sobre Puzzle necesite invocar a servicios de negocio ubicados en otro lugar, debería implementarse el acceso a dichos servicios. Dentro de este ámbito, actualmente Puzzle no establece ninguna infraestructura que facilite de algún modo la creación de estos proxies.

- **Capa de Acceso a Datos**: Puzzle cuenta con un modelo de acceso a datos basado en DAOs que soporta múltiples orígenes de datos, concretamente BD, LDAP y XML). El objetivo es dotar a la información persistente de una representación como objeto.

  El modelo de DAOs de Puzzle se basa en el patrón DAO propuesto en los patrones de diseño J2EE [25] aunque con una salvedad: el modelo planteado impone una relación 1 – 1 entre el DAO de acceso a la fuente de datos y la unidad de almacenamiento de dicha fuente de datos (tabla en BD, nodo en LDAP o fichero XML). Se trata de un enfoque de muy bajo nivel que hace necesario implementar sobre él una capa de abstracción adicional para representar el objeto de negocio. Esta implementación, sin embargo, es responsabilidad de cada uno de los aplicativos que se instancien.

  Se dispone además, como veremos más adelante, de un generador específico para construir estos DAOs a partir de una descripción XML.

Entre estos componentes o capas, cada uno encargado de una tarea, circulan una serie de objetos que transportan información. Dichos objetos, representados por círculos en la Ilustración 10 Framework Puzzle, así como otros componentes adicionales, como los de acceso a datos y activador de servicios son generados automáticamente mediante ficheros descriptores usando una serie de generadores diseñados específicamente para Puzzle y que veremos en los siguientes apartados.

En las siguientes figuras se puede ver la jerarquía de componentes y clases del framework.

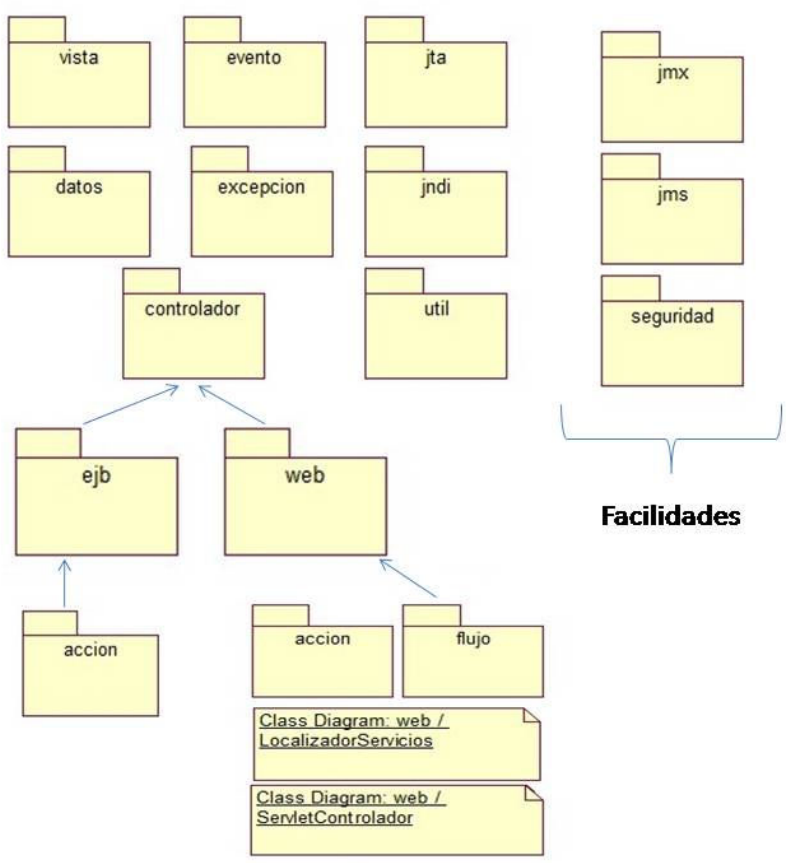

Ilustración 15 Diagrama de componentes de Puzzle

Como se puede ver, existe una agrupación entre los componentes asociados a facilidades, a servicios y herramientas y al propio núcleo del framework.

El diagrama de clases general es el que se ilustra en la siguiente figura:

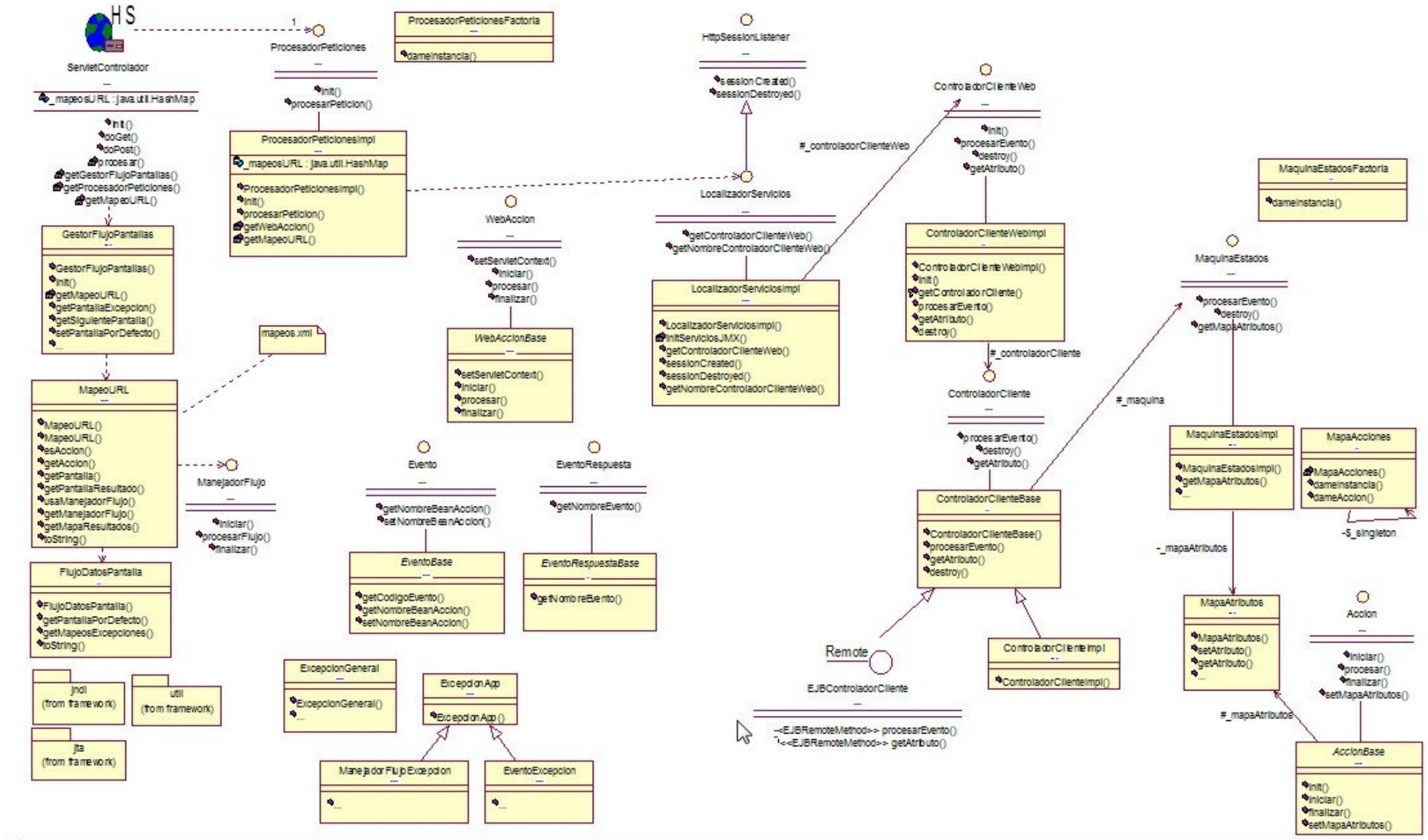

Ilustración 16 Diagrama general de clases de Framework Puzzle

Podemos observar en la ilustración anterior una foto estática de los componentes que intervienen en una aplicación web basada en Puzzle, en particular, una visión general de los componentes que componen el núcleo del framework. La relación o invocación dinámica de los diferentes componentes ilustrados se establecen en el diagrama de secuencia que se describe en el apartado siguiente.

Las aplicaciones web desarrolladas usando Puzzle deben implementar los elementos no proporcionados por el FrameWork de Puzzle, aportando así la funcionalidad propia de la aplicación. En este sentido, vamos a ir descubriendo los elementos que componen Puzzle tomando como guía **el Ciclo de funcionamiento de las aplicaciones basadas en Puzzle.**

5.1 Ciclo de funcionamiento de las aplicaciones Puzzle

Las aplicaciones web basadas en Puzzle deben implementar los elementos no proporcionados por el FrameWork, aportando así la funcionalidad propia de la aplicación.

A la hora de estudiar el funcionamiento de las aplicaciones basadas en Puzzle, es necesario distinguir dos conceptos muy importantes:

- **Acción**: Se entiende por acción una petición realizada por el cliente sobre la aplicación invocando a una URL. Dicha acción puede llevar asociada o no una modificación del modelo. La definición de las acciones es un punto primordial del diseño de la aplicación, y se explicará posteriormente.

- **Pantalla**: Se entiende por pantalla el conjunto de datos devueltos por pantalla al usuario como resultado de una acción realizada por el cliente.

La distinción de estos dos conceptos permite tratarlos por separado, lo cual permite la posibilidad de que a una acción hecha sobre la aplicación, ésta responda con una u otra pantalla en función del resultado de la acción.

El diagrama de flujo de las acciones invocadas sobre la aplicación es el que se muestra en la figura:

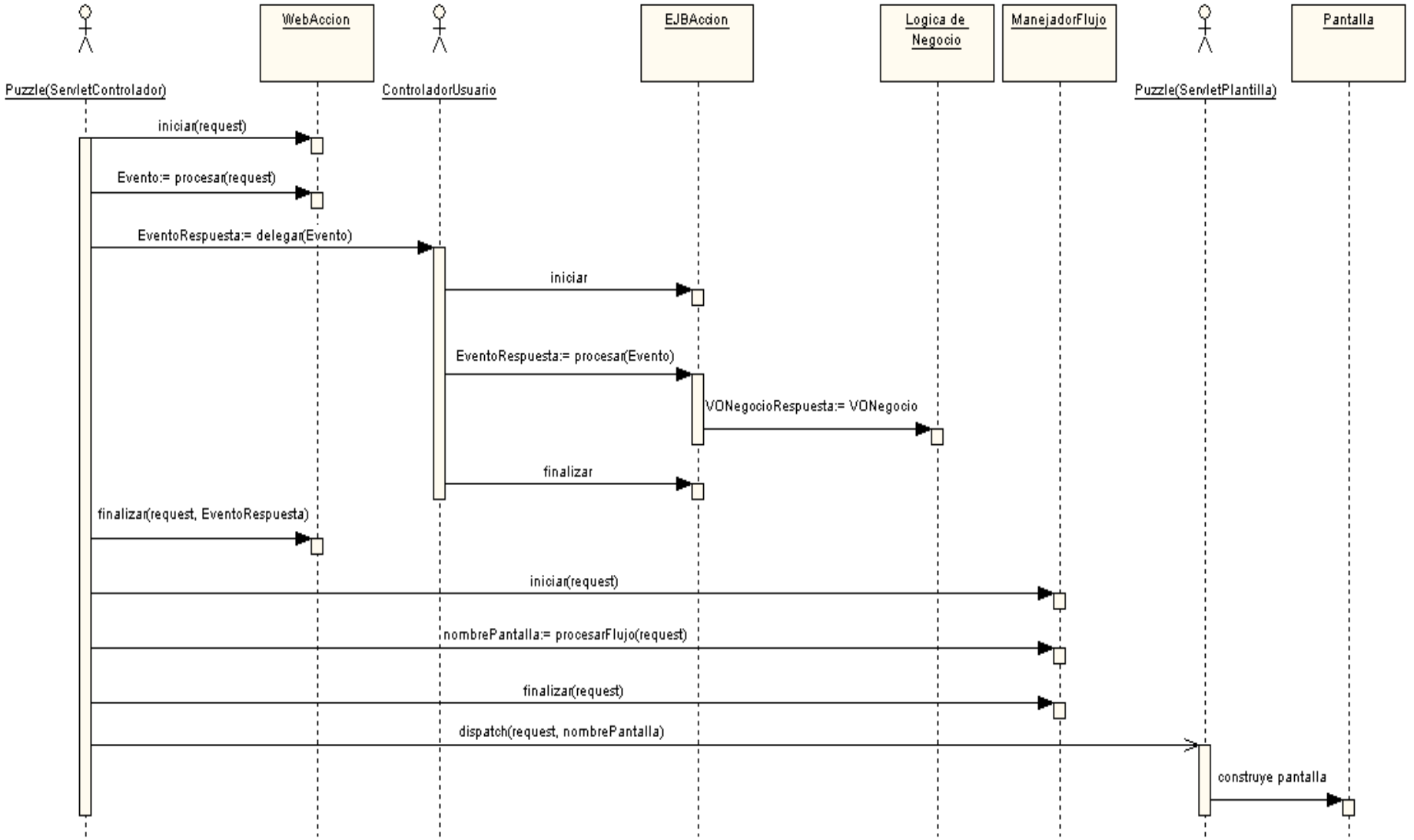

Ilustración 17 Diagrama de Secuencia de una aplicación basada en Puzzle

Cada acción invocada sobre la aplicación por el cliente se compone, básicamente, de la request (petición web) que dicho cliente envía al servidor de aplicaciones y sobre el que se apoya la aplicación. Si esa acción implica una modificación del modelo (posteriormente se estudiará dónde se define si la acción implica modificación del modelo), el propio FrameWork de Puzzle se encarga de tratar esa request, invocando a una clase Java que debe realizar el procesado de la request.

Dicha clase es la *WebAccion*, clase que debe ser implementada específicamente para cada posible acción contra modelo de la aplicación. Si la acción no implica modificación del modelo, el FrameWork se salta todo el procesamiento de lógica de negocio, invocando directamente al *ManejadorFlujo*, tal y como se describe en el apartado 5.2.1 Fichero de acciones.

La finalidad de dicha clase es extraer los datos necesarios de la request enviada por el cliente, mapeándolos a un objeto básico de Puzzle llamado *Evento*.

Evidentemente, la *WebAccion* no es una clase cualquiera, sino que debe presentar unas ciertas características de herencia e implementación de métodos. En concreto, dicha clase debe extender de la clase abstracta proporcionada por Puzzle *tid.puzzle.framework.controlador.web.accion.WebAccionBase,* como se observa en la figura.

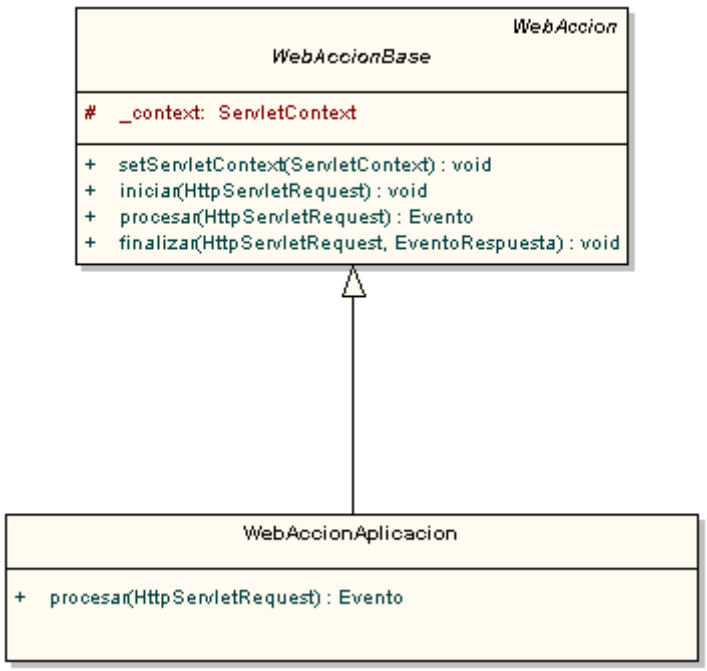

Ilustración 18 Diagrama de Clases de la WebAccion

Lógicamente, se debe plantear una WebAccion distinta para cada posible acción sobre la aplicación. Es en la WebAccion, en el método procesar, donde deben extraerse los parámetros de la request, que son los datos de aplicación enviados por el cliente. A la hora de recogerlos de la request, es altamente conveniente comprobar que los datos cumplen con características que se esperan de ellos, comprobando en concreto longitudes máximas de campos y formatos de fechas y números entre otros. Ese método *procesar* deberá devolver al FrameWork de Puzzle el *Evento* generado a partir de la request.

Como se ha dicho, la WebAccion debe mapear los campos a un objeto Evento. En realidad, lo que se hace es crear objetos Eventos, que heredan de un *EventoBase* abstracto *tid.puzzle.framework.evento.EventoBase,* proporcionado por Puzzle. Dichos eventos, contienen un conjunto de campos que pueden ser de diversos tipos (int, String, long, Date, List, etc...), y los métodos get y set para cada campo.

Además, el evento tiene un campo que indica cuál es la clase que debe ocuparse del procesamiento del Evento. La estructura de clases es la mostrada en la figura:

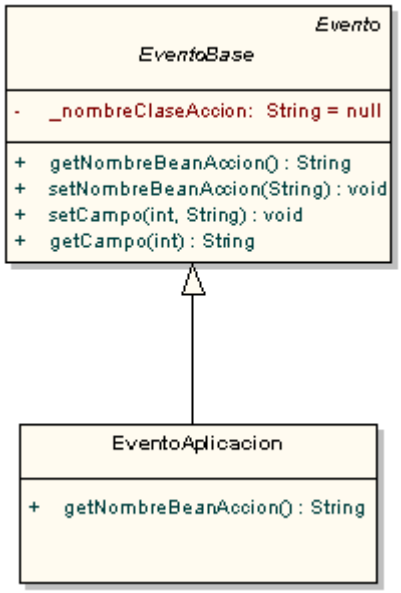

Ilustración 19 Diagrama de Clases del Evento

Posteriormente el FrameWork determina la *EJBAccion* u objeto de Negocio que debe procesar el evento a partir del método *getNombreBeanAccion*. Por lo tanto, un evento está asociado unívocamente a una EJBAccion. De esta manera, el Evento unifica, en un solo objeto, una acción sobre la lógica de negocio y los datos que deben pasarse a la lógica de negocio.

La *EJBAccion* es una clase cuya misión es la invocación de la lógica de negocio que implementa las reglas de negocio propiamente dichas. Su diagrama de clases es el mostrado en la figura, extendiendo de *tid.puzzle.framework.controlador.ejb.accion.AccionBase*.

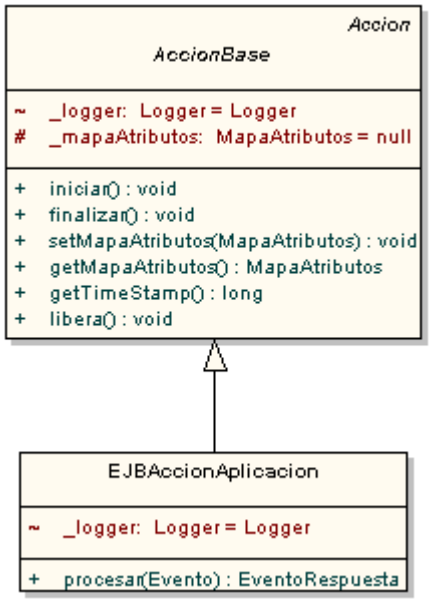

Ilustración 20 Diagrama de Clases de EJBAccion

El método procesar de la EJBAccion recibe el objeto Evento, e invoca a las clases de negocio necesarias para la ejecución de las reglas de negocio aplicables a la acción invocada por el cliente. Dicho método debe devolver un objeto que extienda de *EventoRespuesta*, cuya implementación base del framework es *tid.puzzle.framework.evento.EventoRespuestaBase*, que proporciona los campos y métodos necesarios para devolver el resultado de las reglas de negocio. El diagrama de clases en cuestión es el mostrado en la siguiente figura:

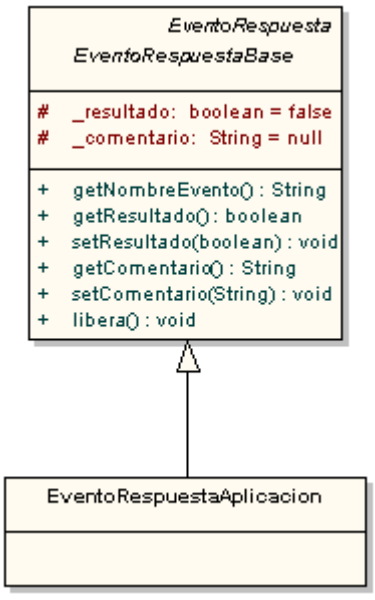

Ilustración 21 Diagrama de Clases del EventoRespuesta

La invocación de las clases de negocio desde la EJBAccion es competencia exclusiva de la aplicación, pudiéndose usar EJBs o simples JavaBeans.

El EventoRespuesta generado desde la EJBAccion es devuelto al FrameWork, que invoca el método finalizar de la WebAcción, pasándole el EventoRespuesta, por si es necesario realizar algún tipo de procesamiento posterior a la lógica de negocio (Véase WebAccion).

Una vez realizada la acción sobre el modelo, se llega al mismo caso que si la acción invocada sobre la aplicación no afectará a modelo, ya que en ambos casos se va a atacar a una clase llamada *ManejadorFlujo* (clase que deberá ser implementada para cada acción del sistema). El cometido de esta clase es decidir que pantalla debe mostrarse al usuario como respuesta a la acción invocada por éste sobre la aplicación.

Asimismo, debe cargar los datos necesarios para la generación de la pantalla que debe mostrarse. El *ManejadorFlujo* debe extender de *tid.puzzle.framework.controlador.web.flujo.ManejadorFlujoBase*, e implementar el método procesar, según se muestra en la figura:

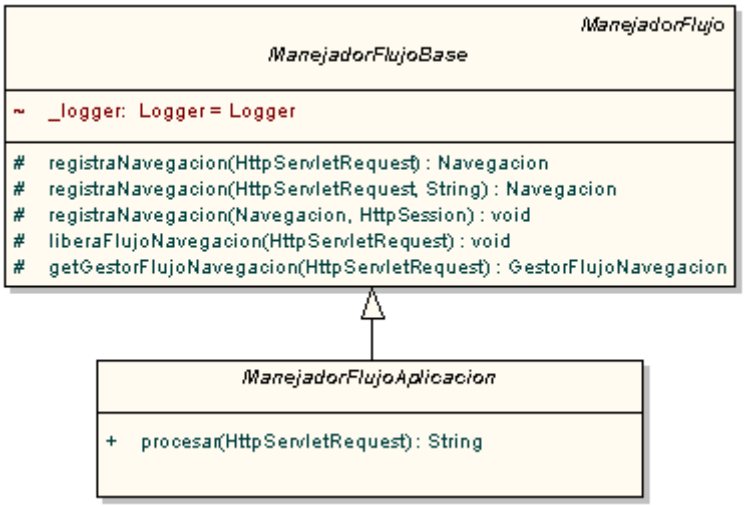

Ilustración 22 Diagrama de Clases del ManejadorFlujo

El manejador consulta al Modelo los datos necesarios para realizar la presentación al usuario, y devuelve al FrameWork una cadena, que contiene el nombre de la pantalla que debe invocarse.

A partir del nombre de la pantalla destino, Puzzle es capaz de generar una respuesta al usuario, invocando a las JSPs que componen la pantalla resultado propiamente dicha. La asociación de una pantalla con una serie de JSP es también un punto esencial del diseño de la aplicación, y se explica a continuación.

5.2 Ficheros de configuración

Este apartado describe los ficheros de configuración necesarios para la comunicación entre las diferentes capas propuestas por el patrón MVC y la implementación de Puzzle.

5.2.1 Ficheros de Acciones y definiciones de pantallas

La asociación de una acción contra la aplicación con una WebAccion y/o ManejadorFlujo concreto por un lado, y la correspondencia entre una pantalla invocable desde un ManejadorFlujo y la JSP que compondrá la respuesta al usuario, se establecen en diversos ficheros descriptores de toda aplicación Puzzle.

Dichos ficheros tiene formato XML.

5.2.1.1 Ficheros de Acciones

El primero de ellos se llama ***mapeos.xml***, y define la asociación entre la URL invocada por el usuario y la WebAcción y/o ManejadorFlujo que debe procesar la petición.

Este fichero es único para cada aplicación. El descriptor de cada URL tiene el siguiente formato:

```
<mapeo_url      url="peticion.accion"
                pantalla="destino.pantalla"
                esAccion="true"
                usaManejadorFlujo="true">
        <clase_accion>
                tid.aplicacion.controlador.web.accion.WebAccionAplicacion
        </clase_accion>
        <manejador_flujo clase="tid.aplicacion.controlador.web.flujo.ManejadorFlujoAplicacion">
                <resultado_manejador    resultado="correcto"
                                        pantalla="destino1.pantalla"/>
                <resultado_manejador    resultado="error"
                                        pantalla="destino2.pantalla"/>
        </manejador_flujo>
</mapeo_url>
```

Ilustración 23 Ejemplo de fichero de Acciones

Donde el significado de los parámetros es el siguiente:

- **mapeo_url**: entidad que representa una posible acción a invocar sobre la aplicación

- **url**: recurso al que se accede en la petición http(s) dentro del contexto de la aplicación.
- **pantalla**: pantalla destino en caso de no usarse manejador de flujo.
- **esAcción**: valor booleano que indica si la petición debe ser procesada por una WebAccion (false implica forward)
- **usaManejadorFlujo**: valor booleano que indica si es necesario un ManejadorFlujo que determine la pantalla destino.
- **clase_accion**: en caso de usarse una WebAcción, el nombre completo de la WebAccion.
- **manejador_flujo**: entidad que representa un ManejadorFlujo.
  - **clase**: el nombre completo del ManejadorFlujo
  - **resultado_manejador**: entidad que representa un posible resultado devuelto por el ManejadorFlujo. Debe haber tantos como posibles resultados pueda devolver el ManejadorFlujo.
    - **resultado**: resultado devuelto por el método procesar del ManejadorFlujo.
    - **pantalla**: nombre de la pantalla asociada al resultado devuelto por el ManejadorFlujo.

Además de las definiciones de acciones, el fichero mapeos.xml debe contener las posibles pantallas de destino que se corresponden con las excepciones que se produzcan durante el procesamiento de la request. Dichos descriptores tienen el siguiente formato:

```
<mapeo_excepcion clase="tid.puzzle.framework.controlador.web.MapeoNoEncontradoExcepcion"

                 pantalla="error404.pantalla"/>
```

Ilustración 24 Ejemplo de fichero de Acciones (excepciones)

Este fichero debe estar localizado en la carpeta ***/WEB-INF/*** de la aplicación web.

5.2.1.2 Ficheros de Pantallas

Las definiciones de las pantallas se establecen en diversos ficheros, cuyos nombres tienen el siguiente aspecto: ***defpantallas_XXX.xml***. La razón de ello es que cada fichero descriptor de pantallas lleva asociada una plantilla. Si en una aplicación web es necesario el uso de más de una plantilla, será necesario usar tantos ficheros defpantallas_XXXX.xml como plantillas sean necesarias. Posteriormente se explicará cómo se asocian cada tipo de pantalla con los nombres de pantallas destino de *mapeos.xml*.

En general los descriptores de las pantallas tienen el siguiente aspecto:

```
<pantalla extends=”nombrePantallaPadre”>
        <nombre>nombrePantalla</nombre>
        <parámetro      clave=”clave”
                        valor =”valor”
                        inclusion_directa=”true”/>
</pantalla>
```

Ilustración 25 Ejemplo de fichero de definición de Pantallas

Donde el significado de los parámetros es el siguiente:

- **pantalla**: entidad que representa una pantalla
  - **extends**: nombre de la pantalla de la cual hereda sus parámetros. Es posible implementar herencia entre pantallas, donde las pantallas hijas heredan los parámetros de la pantalla padre, pudiendo sobrescribirlos. La pantalla padre **debe** definirse antes que la pantalla hija.
  - **nombre**: nombre de la pantalla. Se corresponde con el atributo pantalla de una entidad resultado definida en el mapeos.xml en recurso al que se accede en la petición http dentro del contexto de la aplicación.
  - **parámetro**: pareja clave-valor, que representa parámetros a pasar a la JSP. Lleva asociado un atributo llamado **inclusión_directa**, que es un valor booleano cuyo cometido es determinar si el valor del atributo debe ser procesado o simplemente insertado al usarse.

La generación de las pantallas en Puzzle se basa en el uso de una plantilla, formada por una JSP, a la que se le pasan los parámetros declarados en la definición de la pantalla (de esto se ocupa el framework, donde sólo es necesario crear la plantilla y declarar las propiedades en defpantallas_XXX.xml). Es la propia JSP de plantilla la que debe ocuparse, en función de los parámetros, de hacer las inclusiones de JSPs que sean necesarias. Esta plantilla debe ser construida para cada fichero defpantallas_XXX.xml.

Por tanto, además de las definiciones de pantallas, cada fichero defpantallas_XXX.xml debe definir una entidad que representa la plantilla a usar para generar la pantalla. La entidad a crear tiene el siguiente formato:

```
<plantilla>/plantilla.jsp</plantilla>
```

Ilustración 26 Ejemplo de fichero de definición de Pantallas (Plantillas)

Estos ficheros deben estar localizados en la carpeta ***/WEB-INF/*** de la aplicación web.

5.3 Elementos singulares de la aplicación basada en Puzzle

Además de los elementos descritos en el apartado anterior 5.1 Ciclo de funcionamiento de las aplicaciones Puzzle, que podrían denominarse como 'múltiples', esto es, al existir tantos elementos como acciones en la aplicación, existen, en el desarrollo de aplicaciones Puzzle, la intervención de una serie de elementos singulares.

Dichos elementos son singulares en el sentido de que solo debe definirse una entidad de cada uno de ellos. Los elementos en cuestión son: el ***LocalizadorServicios***, el ***ControladorCliente*** y el ***SerlvetSessionListener*** que corresponden, según el MVC, a la capa controlador.

Por otro lado, su ciclo de vida es distinto, ya que están vinculadas a la sesión web (caso del LocalizadorServicios ó del ControladorCliente) o a la aplicación (SerlvetSessionListener), siendo tan solo preciso para su uso extenderlos y definirlos en el descriptor de la aplicación web (*web.xml*), que veremos más adelante, en el caso de los dos primeros elementos, o simplemente definirlos en el descriptor en el caso del SerlvetSessionListener.

De estos elementos, los dos primeros son instanciados tantas veces como sesiones se creen en la aplicación, y su vida termina cuando se destruye la sesión web.

El SerlvetSessionListener se arranca con la primera petición a la aplicación, y su vida termina cuando se cierra la aplicación.

Estos elementos son proporcionados por Puzzle, pero algunos de ellos pueden ser extendidos para poder ser usados. Además, en función de la versión de Servlets que se deseé usar (2.2 ó 2.3) es necesario utilizar unos u otros.

5.3.1 Clase LocalizadorServicios

El LocalizadorServicios es un componente de la capa Web de Puzzle. Este objeto se instancia cada vez que un usuario inicia una sesión en la aplicación, y su cometido es arrancar los servicios necesarios para el funcionamiento de la aplicación, instanciar e inicializar el *ControladorClienteWeb* y el *GestorFlujoNavegación* para cada sesión que se inicie en la aplicación.

El LocalizadorServicios es un componente que implementa la interfaz *HttpSessionListener*, por lo que exige el uso de la versión 2.3 o superior del API de los Servlets como se describe en los requisitos software.

En caso de no poder usarse esta versión del API, es decir, si tenemos que usar la versión anterior 2.2, será necesario otro componente Puzzle que *"emule"* el funcionamiento del LocalizadorServicios. Ese componentes es el ServletSessionListener (Véase el apartado 5.3.3. Clase ServletSessionListener). El LocalizadorServicios debe definirse como un listener en el descriptor de la aplicación web (Véase 5.4 Descriptor de la aplicación web).

Para usar este componente, es necesario extenderlo para indicarle a Puzzle cuál es el objeto que ejercerá el papel del ControladorClienteWeb. Por ello, la implementación habitual de este componente se limita a sobrecargar el método *getNombreControladorClienteWeb(),* devolviendo el nombre del ControladorClienteWeb usado en la aplicación.

```
package tid.paress.controlador.web;

// Puzzle

import tid.puzzle.framework.controlador.web.LocalizadorServiciosImpl;

public class LocalizadorServiciosParess extends LocalizadorServiciosImpl

{

public String getNombreControladorClienteWeb()

{

        return new String("tid.paress.controlador.web.ControladorUsuarioWeb");

}

}
```

Ilustración 27 Ejemplo de fuente LocalizadorServicios

También habrá que definir la entrada en el descriptor de la aplicación web (Véase 5.4 Descriptor de la aplicación web).

Es posible también sobrecargar el método *initServicios(HttpSession sesion)* para que se arranquen otros servicios propios de la aplicación, pero entonces es necesario que se haga una llamada al método initServicios(HttpSession sesion) de la clase padre, para inicializar los servicios propios de Puzzle.

Una implementación habitual se ilustra en la siguiente figura:

```
package tid.paress.controlador.web;

// Puzzle

import tid.puzzle.framework.controlador.web.LocalizadorServiciosImpl;

public class LocalizadorServiciosParess extends LocalizadorServiciosImpl

{

public String getNombreControladorClienteWeb()

{

return new String("tid.paress.controlador.web.ControladorUsuarioWeb");

}

protected void initServicios(HttpSession sesion)

{

        super.initServicios(sesion);

        // Se inician los servicios de aplicación

        ...

        // Fin de la inicialización de servicios de aplicación.

}}
```

Ilustración 28 Ejemplo de fuente LocalizadorServicios (initServicios)

Es necesario tener en cuenta que el LocalizadorServicios es instanciado cada vez que se abre una nueva sesión. Por ello, a la hora de arrancar servicios de aplicación que son comunes para todos los usuarios, es necesario comprobar previamente que no han sido arrancados y así evitar conflictos.

5.3.2 Clase ControladorCliente

El ControladorCliente es el elemento que proporciona el acceso a la capa de lógica de negocio. De esta forma Puzzle hace transparente la invocación a la capa de negocio, donde para ello, solo es necesario instanciar el evento adecuado y pasárselo al ControladorCliente para su procesamiento. Este componente se instancia para cada sesión iniciada en la aplicación.

El ControladorCliente es un elemento complejo; éste se divide, básicamente, en dos elementos: el ControladorClienteWeb, y el ControladorCliente propiamente dicho (en adelante, el ControladorClienteNegocio). A su vez, Puzzle ofrece una implementación del ControladorClienteNegocio en forma de JavaBean, que puede usarse tal cual, o usar una envoltura EJB del ControladorClienteNegocio, si el entorno de funcionamiento de la aplicación es un servidor de aplicaciones.

En realidad, el ControladorClienteWeb es una envoltura del ControladorClienteNegocio. Cuando se instancia el ControladorClienteWeb, éste debe instanciar el ControladorClienteNegocio, de tal forma que todo acceso a la lógica de negocio se hace a través del ControladorClienteWeb, que debe invocar al ControladorClienteNegocio pasándole el evento tal y como se ilustra en la siguiente figura:

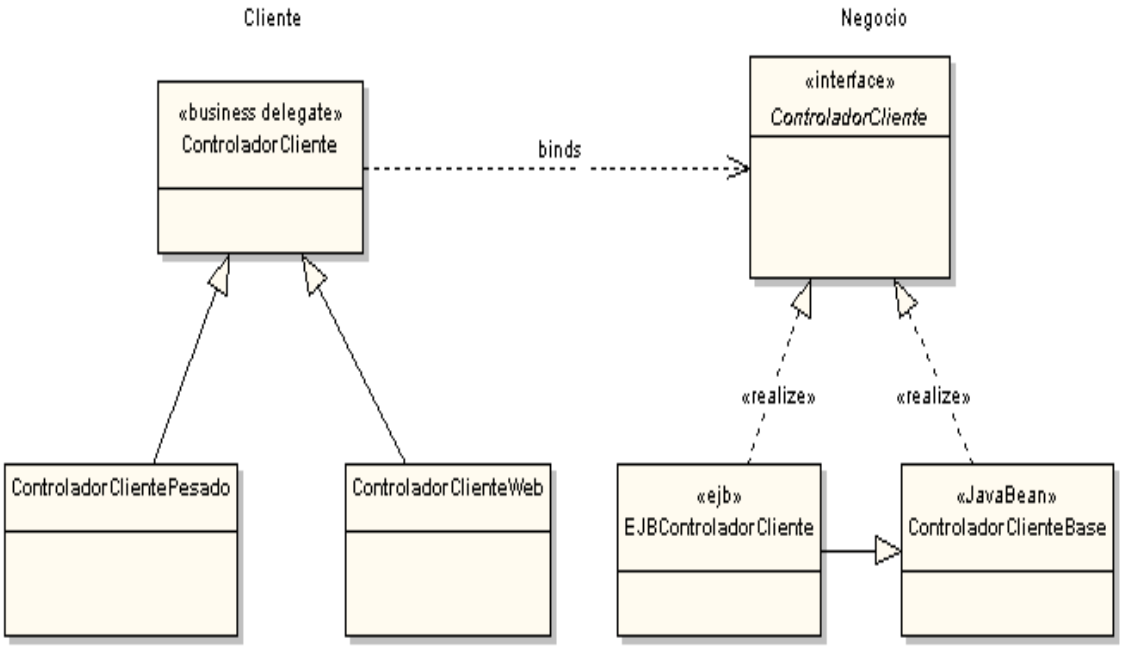

Ilustración 29 Modelo de acceso a la capa de Negocio

Podemos observar un elemento no mencionado, el ControladorClientePesado. Una particularidad de Puzzle es que es posible utilizar el framework sin su capa Web, diseñando una nueva envoltura que permita el acceso al ControladorCliente desde

aplicaciones pesadas (Java Stand-Alone) aunque ese componente no está desarrollado en esta versión.

Puesto que la forma de invocación, e incluso el propio *ControladorClienteNegocio* puede ser distinto en función de la aplicación y del entorno de trabajo (Uso o no de EJB's), es necesario extender la clase ControladorClienteWebBase para que realice la invocación del ControladorClienteNegocio el cual se vaya a usar finalmente en la aplicación instanciada.

Para ello, se debe extender de ControladorClienteWebBase e implementar el método *getControladorCliente(),* que devuelve el ControladorCliente a usar.

La relación ControladorClienteWeb←→ControladorClienteNegocio, que forman el ControladorCliente, debe ser persistente a lo largo de la sesión, ya que la invocación de la lógica de negocio siempre se hace a través del ControladorClienteWeb.

Por tanto, dado que el ControladorClienteWeb es persistente (está ligado a la sesión del usuario), se sugiere que se mantenga una referencia al ControladorClienteNegocio, devolviendo siempre el ControladorClienteNegocio asociado. Antes de devolverlo, se debería de comprobar la validez del ControladorClienteNegocio.

Por ejemplo, si se trabaja en un entorno con EJBs, una manera habitual de extender ControladorClienteWebBase es la siguiente:

```
package tid.paress.controlador.web;

// Jdk

import java.util.Properties;

import javax.naming.Context;

import javax.naming.InitialContext;

import javax.rmi.PortableRemoteObject;

// Trazas

import org.apache.log4j.Logger;

// Puzzle

import tid.puzzle.framework.controlador.web.ControladorClienteWeb;

import tid.puzzle.framework.controlador.web.ControladorClienteWebBase;

import tid.puzzle.framework.controlador.ejb.ControladorCliente;

// paress

import tid.paress.controlador.ejb.EJBControladorUsuarioHome;

import tid.paress.controlador.ejb.EJBControladorUsuario;

public class ControladorUsuarioWeb extends ControladorClienteWebBase

                                   implements ControladorClienteWeb
```

```
{
	/**
	* Sistema de trazas
	*/
	static Logger _logger = Logger.getLogger(ControladorUsuarioWeb.class);
	/**
	* ControladorEJB (referencia 1-1)
	*/
	protected EJBControladorUsuario _controladorEJB = null;
	/**
	* Constructor
	*/
	public ControladorUsuarioWeb()
	{
		super();
	}
	/**
	* Recupera la parte Servidor del componente Controlador
	*/
	protected ControladorCliente getControladorCliente()
	{
		//comprobar si la referencia del EJB es válida. No vale con la comprobación
		//del nulo ya que la referencia local no se inicializa (igual a nulo) cuando el
		//contenedor efectúa el remove del ejb
		_logger.debug("Comprobando el estado Capa Servidor Controlador");
		try
		{
			//realizamos una llamada remota para comprobar si aun es valida
			_controladorEJB.getAtributo("");
			_logger.debug("Activo");
		}
		catch (Exception e)
		{
			//_logger.debug("Se ha perdido la referencia al EJB", e);
```

```
                _controladorEJB = null;
        }
        _logger.debug("Recuperando Capa Servidor Controlador");
        if (_controladorEJB == null)
        {
                Properties env = null;
                Context ctx = null;
                EJBControladorUsuarioHome home = null;
                try
                {
                        _logger.debug("Instanciando el controlador EJB");
                        // Recuperamos la capa Servidor del ControladorCliente
                        ctx = new InitialContext();
                        Object objHome = ctx.lookup("EJBControladorUsuario");
                        home= (EJBControladorUsuarioHome)
                        PortableRemoteObject.narrow(objHome,
                        EJBControladorUsuarioHome.class);
                        _controladorEJB = home.create();
                        _logger.debug("Tenemos el EJB:" + _controladorEJB);
                }
                catch (Exception ce)
                {
                        _logger.error("Error Recuperando ControladorCliente EJB", ce);
                        throw new RuntimeException(ce.getMessage());
                }
                finally
                {
                        if(env != null)
                        {
                                env.clear();
                                env = null;
                        }
                        ctx = null;
                        home = null;
                }
```

```
                }
                return _controladorEJB;
        } // getControladorEJBCliente
        /**
        * Finalización del ControladorCliente
        */
        public void destroy()
        {
                try
                {
                        super.destroy();
                        _logger.debug("Eliminando EJB Session Controlador");
                        try
                        {
                                //eliminamos
                                _controladorEJB.remove();
                        }
                        catch (Exception eremote)
                        {}
                        _controladorEJB = null;
                }
                catch (Exception e)
                {}
        } // destroy
} // class
```

Ilustración 30 Ejemplo de fuente ControladorClienteWeb (EJB based)

Si el entorno de trabajo no tiene EJBs, se puede utilizar directamente el JavaBean ControladorClienteBean, tal y como se ilustra en el siguiente ejemplo:

```
package tid.paress.controlador.web;

// Jdk

// Trazas

import org.apache.log4j.Logger;

// Puzzle

import tid.puzzle.framework.controlador.web.ControladorClienteWeb;

import tid.puzzle.framework.controlador.web.ControladorClienteWebBase;

import tid.puzzle.framework.controlador.ejb.ControladorCliente;

// paress

import tid.paress.controlador.ejb.ControladorUsuarioBean;

public class ControladorUsuarioWebTomcat extends ControladorClienteWebBase
                                        implements ControladorClienteWeb
{
        /**
        * Sistema de trazas
        */
        static Logger _logger = Logger.getLogger(ControladorUsuarioWeb.class);
        /**
        * Controlador (referencia 1-1)
        */
        protected ControladorCliente _controlador = null;
        public ControladorUsuarioWeb()
        {
                super();
        }
        /**
        * Recupera la parte Servidor del componente Controlador
        */
        protected ControladorCliente getControladorCliente()
        {
//comprobar si la referencia del EJB es válida. No vale con la comprobación del
```

```
//nulo ya que la referencia local no se inicializa (igual a nulo) cuando el //contenedor efectúa el remove del ejb
                    _logger.debug("Comprobando el estado Capa Servidor Controlador");
                    try
                    {
                              //realizamos una llamada remota para comprobar si aun es valida
                              _ controlador.getAtributo("");
                              _logger.debug("Activo");
                    }
                    catch (Exception e)
                    {
                              //_logger.debug("Se ha perdido la referencia al EJB", e);
                              _ controlador = null;
                    }
                    _logger.debug("Recuperando Capa Servidor Controlador");
                    if (_controlador == null)
                    {
                              _ controlador = new ControladorUsuarioBean();
                    }
                    return _ controlador;
          } // getControladorCliente
          /**
          * Finalización del ControladorCliente
          */
          public void destroy()
          {
                    try
                    {
                              super.destroy();
                              _logger.debug("Eliminando Session Controlador");
                              try
                              {
                                        //eliminamos
                                        _ controlador.destroy();
                              }
```

```
                    catch (Exception eremote){}

                    _ controlador = null;

                }

                catch (Exception e){}

        } // destroy

} // class
```

Ilustración 31 Ejemplo de fuente ControladorClienteWeb (no EJB)

Es necesario tener siempre en mente que el ControladorClienteWeb es único para cada usuario, es decir, hay una instancia del ControladorClienteWeb (y por tanto del ControladorClienteNegocio, ya sea EJB o JavaBean) asociada a cada sesión web.

5.3.3 Clase ServletSessionListener

En aquellas aplicaciones que vayan a ejecutarse en un entorno que no soporte el API de Servlets 2.3, sino tan solo 2.2, no se pueden usar los listener, ya que ésta versión no los soporta. En cualquier caso, Puzzle se adapta a esta circunstancia, y ofrece la posibilidad de usar un Servlet que proporciona el funcionamiento del LocalizadorServicios.

Este servlet es el *ServletSessionListener*, para el que no es preciso desarrollo alguno, sino tan solo indicar y añadir la entrada correspondiente en el descriptor de la aplicación web como se describe en el siguiente apartado.

Este sistema permite usar Puzzle con el API de Servlets 2.2, pero a cambio presenta dos inconvenientes:

- Un mayor consumo de recursos (ya que toda petición pasa previamente por el ServletSessionListener),
- ya no se puedan arrancar servicios de aplicación, como se hacía con el LocalizadorRecursos.

5.4 Descriptores de aplicación

Una aplicación web basada en Puzzle se despliega igual que todas las demás aplicaciones web, y necesita por tanto, los ficheros descriptores recogidos en los diversos estándares. Los descriptores necesarios, así como el empaquetado de la aplicación dependerá de varios factores (si hay o no EJB's, si se despliega en un servidor web o en un servidor de aplicaciones, etc).

Si el módulo a desplegar es una simple aplicación web, sin EJBs, o si la parte web de la aplicación se va a desplegar por separado en un servidor web, es necesario definir el descriptor de la aplicación web: ***web.xml.***

Si lo que se despliega es un módulo EJB, sin parte web, o es la parte EJB de una aplicación que se va a desplegar en un servidor de aplicaciones, es necesario definir el

descriptor del módulo EJB: ***ejb-jar.xml***. En este caso, es necesario incluir la EJBControladorUsuario, además de todas las EJB's usadas en la aplicación.

Por último, si se va a desplegar en un servidor de aplicaciones una aplicación con parte web y parte EJB, es necesario definir el fichero descriptor de la aplicación: ***application.xml.***

5.4.1 Descriptor de la aplicación web

El descriptor de la aplicación web.xml que se debe generar para una aplicación basada en Puzzle varía dependiendo de si la versión del API servlet a usar es la 2.3 o la 2.2. Las DTDs correspondientes pueden encontrarse en:

- http://java.sun.com/dtd/web-app_2_3.dtd o
- http://java.sun.com/j2ee/dtds/web-app_2_2.dtd

En caso de usar la versión 2.3, Puzzle exige que se definan al menos los siguientes elementos:

- **listener**: Se debe definir el listener, cuya clase será LocalizadorServicios. Para definirlo es necesario insertar la siguiente entidad en el archivo web.xml

```
<listener>
<listener-class>
        tid.paress.controlador.web.LocalizadorServiciosParess
</listener-class>
</listener>
```

Ilustración 32 Web Deployment Descriptor (web.xml) Identificación del Listener

Donde la clase será la creada específicamente en la aplicación extendiendo de la proporcionada por Puzzle (Véase el apartado 5.2 LocalizadorServicios).

- **servlet**: Puzzle exige que se definan dos servlet, ambos proporcionados por el framework. El primero de ellos es el *ServletControlador*, que atenderá las peticiones realizadas por el usuario. Su definición es la siguiente:

```
<servlet>
```

```
<servlet-name>ServletControlador</servlet-name>
<display-name>ServletControlador</display-name>
<description>Controlador Frontal del Sistema entorno Web</description>
<servlet-class>tid.puzzle.framework.controlador.web.ServletControlador</servlet-class>
</servlet>
```

Ilustración 33 Web Deployment Descriptor (web.xml) Identificación del Servlet Controlador

Este servlet no necesita ningún parámetro. El segundo servlet, es el *ServletPlantilla*, cuya misión es redirigir la salida del ServletControlador a la pantalla de destino. Se deben definir tantos ServletPlantilla como plantillas haya en la aplicación. El formato del descriptor es el siguiente:

```
<servlet>
<servlet-name>ServletPantalla</servlet-name>
<display-name>ServletPlantilla</display-name>
<servlet-class>tid.puzzle.framework.vista.plantilla.ServletPlantilla</servlet-class>
<init-param>
        <param-name>lenguaje_por_defecto</param-name>
        <param-value>sp</param-value>
</init-param>
</servlet>
```

Ilustración 34 Web Deployment Descriptor (web.xml) Identificación del Servlet Controlador Pantallas

Cada ServletPlantilla definido está asociado a un lenguaje determinado, dado por el parámetro *lenguaje_por_defecto*. Esta asociación se extiende al fichero descriptor de pantallas, de tal forma que cada ServletPlantilla tiene definido un fichero defpantallas_XXXX.xml, en donde el sufijo XXXX tiene el valor del parámetro lenguaje_por_defecto. De esta forma, para el servlet definido en el ejemplo, el fichero descriptor de pantallas es *defpantallas_sp.xml*. El nombre del Servlet debe ser distinto para cada ServletPlantilla.

Conseguimos por tanto, la posibilidad de definir diversos servlet de presentación, asociados cada uno a una plantilla, que tienen determinadas pantallas.

- **servlet-mapping**: Para cada Servlet definido en la aplicación web, es necesario definir que patrones de URL debe atender. Al menos se deben definir dos mapeos. El primero de ellos se refiere al ServletControlador:

```
<servlet-mapping>
<servlet-name>ServletControlador</servlet-name>
<url-pattern>*.accion</url-pattern>
</servlet-mapping>
```

Ilustración 35 Web Deployment Descriptor (web.xml) Servlet Mapping (acciones)

De esta forma, toda petición de la forma ***.acción*** será atendida por el ServletControlador, que utilizará el fichero de acciones (Véase apartado 4.2.1 Fichero de acciones) para procesar la petición.

Para cada ServletPlantilla definido en la aplicación, es necesario definir un mapeo distinto, cuyo patrón sea distinto. Las pantallas de destino definidas en el fichero de acciones deben cumplir uno de dichos patrones, indicado así qué ServletPlantilla debe procesar la pantalla.

```
<servlet-mapping>
<servlet-name> ServletPantalla </servlet-name>
<url-pattern>*.pantalla</url-pattern>
</servlet-mapping>
```

Ilustración 36 Web Deployment Descriptor (web.xml) Servlet Mapping (pantallas)

De esta forma el contenedor Web, cuando recibe una petición que cumple el patrón **.accion*, lo redirige al ServletControlador. Este Servlet procesa la petición, y obtiene una pantalla destino definida en el fichero de acciones, que cumple el patrón **.<lenguaje>,* y que debe estar mapeado a un ServletPlantilla, al que se le hace el dispatch de la petición para que procese la pantalla.

Por tanto, por cada posible plantilla debe definirse un ServletPlantilla, que generará las pantallas destino cuya url cumple el patrón definido en el mapeo. De esta forma, se crea una asociación patrón←→ServletPlantilla←→lenguaje_por_defecto←→defpantallas_XXXX.xml←→plantilla.

Cuando el ServletControlador determina la pantalla destino, se hace el dispatch al ServletPlantilla correspondiente, que generará la pantalla en

base a la plantilla definida y a los parámetros definidos en el fichero de definición de pantallas asociado al ServletPlantilla.

Adicionalmente a los elementos a definir exigidos por Puzzle, habitualmente se definen otros elementos, entre los que podemos destacar los siguientes:

- **session-config**: Este elemento se define para declarar el tiempo máximo que se conserva la sesión si no se reciben peticiones. Ese tiempo se declara en minutos.
- **welcome-file-list:** Indica la url que se ejecuta si se hace una petición a la aplicación sin especificar la acción. Lo normal es que se invoque a una página html que con un enlace a la aplicación.
- **resource-env-ref**: Se utiliza para declarar una referencia a un objeto asociado con un recurso proporcionado por el servidor web. La definición del objeto es propia de cada servidor web/ aplicaciones, debiendo consultar el manual de administrador del servidor en cuestión. Se utiliza habitualmente para declarar el *DataSource.*
- **security-constraint**: Indican restricciones en el acceso a recursos de la aplicación desde el exterior. Además de aquellas restricciones necesarias para la seguridad de la aplicación (uso de roles, recursos protegidos, etc.), es muy conveniente restringir el acceso a los fichero jsp y a los patrones de pantalla, para evitar el acceso directo a dichos recursos, que solo deben ser utilizados por la aplicación Puzzle, vía *dispatch method.*

```
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
[….]
<welcome-file-list>
<welcome-file>index.htm</welcome-file>
</welcome-file-list>
[….]
<resource-env-ref>
        <description>DB Connection</description>
        <resource-env-ref-name>TIDPool</resource-env-ref-name>
        <resource-env-ref-type>javax.sql.DataSource</resource-env-ref-type>
</resource-env-ref>
```

```
[….]
<security-constraint>
<web-resource-collection>
        <web-resource-name>SoloAccesoInterno</web-resource-name>
        <description>Protegemos los recurso internos de acceso externos</description>
        <url-pattern>*.jsp</url-pattern>
        <url-pattern>*.pantalla</url-pattern>
        <http-method>POST</http-method>
                <http-method>GET</http-method>
</web-resource-collection>
<auth-constraint>
        <role-name>CONTAINER</role-name>
</auth-constraint>
</security-constraint>
```

Ilustración 37 Web Deployment Descriptor (web.xml) otras posibles configuraciones

Podemos encontrar un ejemplo completo en el Anexo A: Ejemplo de Web Deployment Descriptor versión 2.3 y en el Anexo B para la versión 2.2, para el caso de usar la versión 2.2 del API de los servlet, ya que, como hemos citado, no se puede usar el LocalizadorServicios, sino que es necesario usar el ServletSessionListenet y que para ello, se define un nuevo servlet:

```
<servlet>
<servlet-name>ServletSessionListener</servlet-name>
<display-name>ServletSessionListener</display-name>
<description>Emulación SessionListener</description>
<servlet-class>tid.puzzle.framework.controlador.web.ServletSessionListener</servlet-class>
        <init-param>
                <param-name>servlet</param-name>
                <param-value>ServletControlador</param-value>
        </init-param>
        <init-param>
                <param-name>controladorUsuarioWeb</param-name>
                <param-value>tid.paress.controlador.web.ControladorUsuarioWeb</param-value>
        </init-param>
</servlet>
```

Ilustración 38 Web Deployment Descriptor (web.xml) Servlet Mapping (pantallas)

La configuración del Servlet se limita a indicarle el servlet que procesará la petición, que siempre será el ServletControlador y la clase del ControladosClienteWeb, al igual que se hacía en la extensión de LocalizadorServicios, pero en vez de vía código, vía descriptor. Es importante prestar atención al nombre de los parámetros, que debe ser exactamente igual que el puesto en el ejemplo, o la aplicación no funcionarán.

Además es necesario cambiar los mapeos, de tal forma que el servlet que atienda las peticiones **.accion* sea el ServletSessionListener, quien posteriormente delegará en el ServletControlador (cuya definición debe mantenerse).

5.5 Generadores de Puzzle

Puzzle proporciona una herramienta para la generación automática de código. Esta herramienta es una librería Java (archivo jar) que contiene todas las clases necesarias para la generación de ciertos componentes. A la hora de utilizarlo, es necesario tener en cuenta ciertos aspectos:

- El generador asume que las fuentes del proyecto está situadas en un directorio fuentes, que cuelga del directorio raíz del proyecto, y allí dejará las clases generadas.
- El package de las clases generadas empiezan siempre de la siguiente forma: ***tid.<nombre_aplicación>,*** a partir de la cual se crean diversos packages en función de la clase generada.

El Objetivo del generador es conjugar flexibilidad con prestaciones y productividad. Estos objetivos se consiguen mediante la generación automática de aquellos objetos que pueden "deducirse" a través de una simple descripción. En este sentido, la descripción se define en un fichero XML que contiene, en líneas generales, los descriptores de los elementos a generar. Dicho descriptor es propietario del tipo de objeto a generar; Así por ejemplo, el descriptor del DAO es distinto del de la WebAcción, y de todos los demás.

Para cada posible objeto a generar, Puzzle proporciona un generador (clase principal) distinto, que procesará los objetos del tipo que correspondan. Así, la clase GeneradorDAO solo procesará los descriptores de tipo DAO que encuentre en el fichero XML de descriptores. Por tanto, el generador permite definir todos los descriptores en un mismo archivo o en varios, sin afectar para nada al funcionamiento.

El fichero XML de generación debe presentar un formato similar al siguiente:

```
<?xml version="1.0"?>
<Proyecto>
        <DAO...>
        [.....]
</DAO>
<Evento...>
[.....]
</Evento>
<AccionWeb... />
</Proyecto>
```

Ilustración 39 Fichero de descripción en XML para la Generación de Componentes Puzzle

La herramienta de generación auxiliar es bastante básica y el código generado es, en ciertos casos, perfectamente operativo y en otros, un simple esqueleto sobre el que completar la programación.

Los elementos que se pueden generar con las herramientas, y que se describen en los siguientes apartados, se ilustra en la siguiente figura:

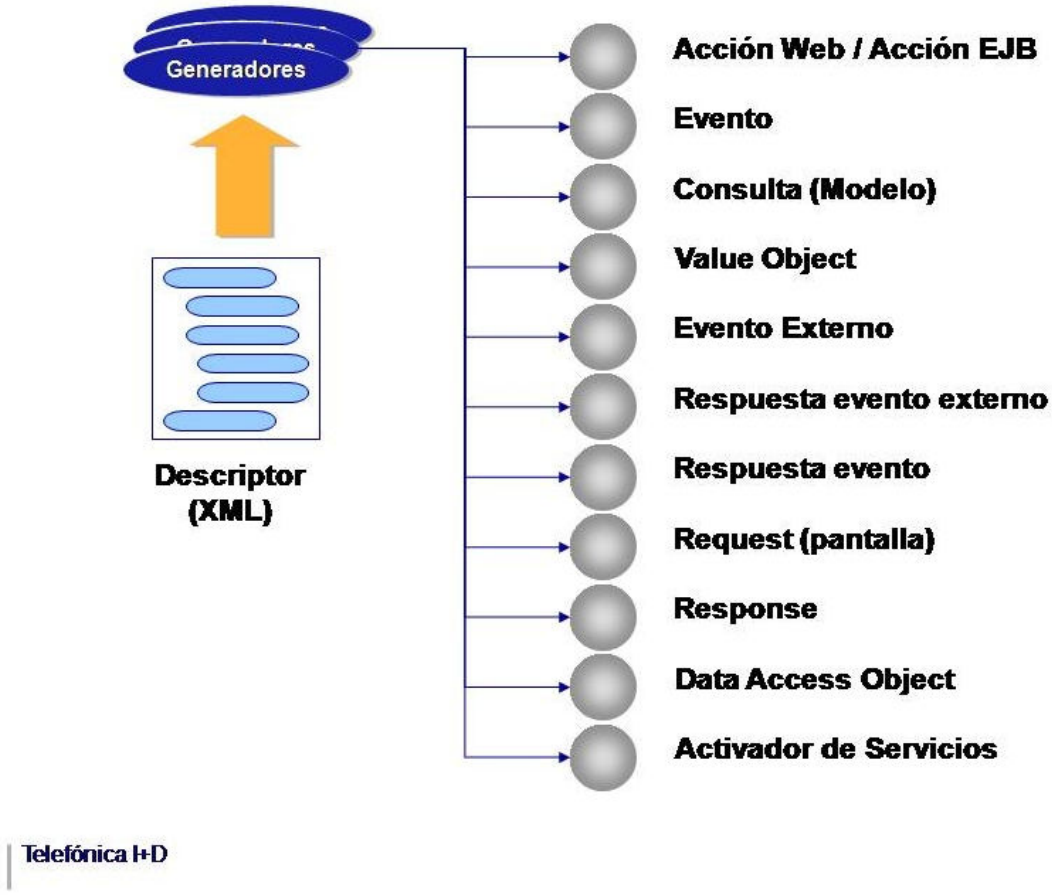

Ilustración 40 Herramientas y componentes generados de Puzzle

En los siguientes apartados, se explican los distintos descriptores a usar.

5.5.1 Generador de Acciones Web: WebAccion

Como se ha comentado anteriormente, la WebAccion es la clase encargada de la recogida de datos de la request, y su mapeo a un objeto Evento manejable por el FrameWork de Puzzle, con objeto de ejecutar la acción correspondiente contra la lógica de negocio. Puesto que los datos a recoger de la request serán distintos de una acción a otra, debe hacer una WebAccion distinta para cada posible acción contra la aplicación.

La WebAccion debe implementar un total de tres métodos:

- iniciar(request),
- procesar(request) y
- finalizar(request,EventoRespuesta).

La WebAccionBase proporcionada por Puzzle aporta una implementación vacía de los tres métodos de la WebAccion.

De esos tres métodos, el más importante es el método *procesar(),* que debe generar el Evento a pasar a la Lógica de negocio. El método *iniciar()* rara vez se

sobrecarga, ya que prácticamente no se usa. Su utilidad es la inicialización de la WebAccion cuando sea necesario.

El método *finalizar(request,EventoRespuesta)* si se suele sobrecargar, con objeto de dejar el EventoRespuesta devuelto por la Lógica de negocio en la request, de tal forma que sea accesible por los demás elementos del ciclo de vida de la acción (estos son el ManejadorFlujo y Pantalla). A menudo es útil definir una WebAccionBase abstracta propia de la aplicación, de la que hereden todas las demás WebAccion, que implemente el método finalizar, para copiar el EventoRespuesta en la request.

Un ejemplo de ello sería el siguiente código:

```
package tid.aplicacion.controlador.web.accion;

// JDK

import javax.servlet.http.HttpServletRequest;

// Puzzle

import tid.puzzle.framework.controlador.web.accion.WebAccionBase;

import tid.puzzle.framework.evento.EventoRespuesta;

// Aplicacion

import tid.aplicacion.util.Constantes;

/**

 * <p>Título: WebAccionAplicacionBase</p>

 * <p>Descripción: Esta WebAccion se encarga de implementar el método finalizar,

 * cargando el EventoRespuesta en la request</p>

 * <p>Copyright: Copyright (c) 2004</p>

 * <p>Empresa: Telefónica I+D</p>

 * @author División 2731 - Equipo de PUZZLE

 * @version 1.0

 */

public abstract class WebAccionAplicacionBase extends WebAccionBase

{

        public void finalizar(HttpServletRequest request, EventoRespuesta eventoRespuesta)

        {

                request.setAttribute(Constantes.ATRB_EVENTO_RESPUESTA, eventoRespuesta);

        }

}
```

Ilustración 41 Ejemplo de fuente generado WebAccion

Donde se ha utilizado una clase de constantes para que la clave que identifica el EventoRespuesta en la request sea la misma para todas las acciones.

La recogida de parámetros de la request debe hacer utilizando la clave por la que se identifican estos. Por ello, es útil definir una clase pública estática interna, que habitualmente se llamará CodigosCampoRequest, cuyos miembros son variables de tipo final String. De esta forma, identificado los campos del formulario HTML con las constantes así definidas, la recogida de datos de la request será mucho más sencilla.

El procedimiento habitual de trabajo será el mostrado en el siguiente trozo de código:

```
package tid.aplicacion.controlador.web.accion;

// JDK

import javax.servlet.http.HttpServletRequest;

// Puzzle

import tid.puzzle.framework.controlador.web.accion.WebAccionExcepcion;

// Aplicacion

import tid.aplicacion.util.Constantes;

import tid.aplicacion.evento.EventoCargaDatos;

/**

 * <p>Título: WebAccionCargaDatosBase</p>

 * <p>Descripción: Esta WebAccion se encarga de cargar datos de la request </p>

 * <p>Copyright: Copyright (c) 2004</p>

 * <p>Empresa: Telefónica I+D</p>

 * @author División 2731 - Equipo de PUZZLE

 * @version 1.0

 */

public class WebAccionCargaDatos extends WebAccionAplicacionBase

{

        public static class CodigosCampoRequest

        {

                public static final String NOMBREAPLICACION     = "NOMBREAPLICACION";

        }

        public Evento procesar(HttpServletRequest request)

        {

                EventoCargaDatos evento = new EventoCargaDatos();

                String nombreAplicacion = null;

                nombreAplicacion= request.getParameter(CodigosCampoRequest.NOMBREAPLICACION);
```

```
                if(nombreAplicacion!=null)

                {

                        if (nombreAplicacion.trim().length()>80)

                                throw new WebAccionExcepcion ("Nombre de la aplicación demasiado
                                largo");

                        evento.setNombreAplicacion(nombreAplicacion.trim());

                }

                return evento;

        }

}
```

Ilustración 42 Ejemplo de fuente generado WebAccion (2)

Puesto que Puzzle proporciona controles de presentación para los datos a enviar a la aplicación, con validaciones de longitud y de formato, es conveniente comprobar los datos en el propio cliente antes de enviar el formulario HTML. Sin embargo, no está de más la comprobación de datos en la WebAccion, ya que de esa manera se puede proteger la aplicación de las posibles manipulaciones de la request hechas por algún intruso. Al detectarse en la WebAccion un dato erróneo, ya sea porque falta cuando es obligatorio, o porque la longitud o el formato son incorrectos, debe lanzarse una WebAccionExcepcion, que provocará el cierre de la sesión y el envío de la pantalla de error correspondiente, que debe estar definida en mapeos.xml

Al utilizar los códigos de campo, en la JSP solo deberá introducirse el siguiente elemento:

```
<input type=”text” name=”<%=WebAccionCargaDatos.CodigosCampoRequest.NOMBREAPLICACION%>”
maxLength=”80”/>
```

Ilustración 43 Ejemplo de fuente generado WebAccion y uso (3)

Es necesario realizar en la cabecera de la jsp el import de la clase correspondiente:

```
<%@ page import="tid.aplicacion.controlador.web.WebAccionCargaDatos"%>
```

Ilustración 44 Ejemplo de fuente generado WebAccion y uso (4)

Finalmente, desde la WebAccion se puede acceder al ControladorClienteWeb para ejecutar métodos (principalmente el *getTimeStamp()),* de la siguiente forma:

```
protected ControladorClienteWeb getControladorCliente(HttpServletRequest request)

{

        return
```

```
        (ControladorClienteWeb)
        request.getSession().getAttribute(tid.puzzle.framework.util.Constantes.CONTROLADOR_CLIENTE_WEB);

}
```

Ilustración 45 Ejemplo de fuente generado WebAccion y uso (5)

5.5.1.1 Uso del generador

Como se ha descrito anteriormente, el descriptor de la WebAccion Puzzle proporciona una herramienta para la generación automática de código, en este caso, para el generador de los componentes correspondientes a WebAcciones, donde el fichero descriptor usado para generar los componentes debe tener formato XML como se indica a continuación:

```
<AccionWeb clase="WebAccionCargaDatos" />
```

Ilustración 46 Ejemplo de descriptor XML para la generación de WebAccion

Y donde el atributo ***clase*** indica el nombre de la WebAccion a generar.

La clase del generador de WebAccion es ***tid.puzzle.herramienta.generador.GeneradorWebAccion***, y está contenido en el jar dado.

Los parámetros que se le pasan al generador son los siguientes:

- **-f** : fichero xml de definición de la WebAccion
- **-d** : Directorio destino donde se generará el código
- **-p** : Nombre del proyecto para el que se generarán las clases (influye en el paquete destino)
- **-h** : muestra la ayuda del generador.

Asumiendo que la librería del generador está en una carpeta generación, dentro de la carpeta raíz del proyecto, y que el proyecto se llama aplicación, una línea de comandos válida para invocar al generador podría ser la siguiente:

```
java -cp

generacion/generador-0.9.jar

tid.puzzle.herramienta.generador.GeneradorWebAccion -f generacion/generacionWebAccion.xml -d . -p
aplicacion
```

Ilustración 47 Ejemplo de llamada para la generación de WebAccion

Las WebAccion así generadas se crean siempre en el paquete *tid.aplicacion.controlador.web.accion*. La estructura que generan es la siguiente:

```
package tid.aplicacion.controlador.web.accion;

// Jdk

import javax.servlet.http.HttpServletRequest;

// Trazas

import org.apache.log4j.Logger;

// Puzzle

import tid.puzzle.framework.controlador.web.accion.WebAccionBase;

import tid.puzzle.framework.controlador.web.accion.WebAccionExcepcion;

import tid.puzzle.framework.evento.Evento;

import tid.puzzle.framework.evento.EventoRespuesta;

// Puzzle - Validacion

import tid.puzzle.framework.util.Validador;

public class WebAccionCargaDatos extends WebAccionBase

{

	static Logger _logger = Logger.getLogger(WebAccionCargaDatos.class);

	public Evento procesar (HttpServletRequest request) throws WebAccionExcepcion

	{

		// Aqui tu código

		return null;

	} // procesar

	public void finalizar (HttpServletRequest request, EventoRespuesta eventoRespuesta)

	{

		// Aqui tu código

	}

} // class
```

Ilustración 48 Resultado de la generación de WebAccion

El generador proporciona un esqueleto que ya implementa todos los métodos necesarios. Solo es necesario proporcionarles contenido. Es necesario destacar que por defecto el generador incluye varias facilidades en la WebAccion: las trazas y el Validador. En los siguientes apartados se explican con detalle estas utilidades.

Al esqueleto generado solo es necesario crearle la clase de códigos de campo de la request, generando una constante distinta para cada campo; posteriormente instanciar el Evento de la clase que sea necesaria y cargarlo con los datos de la request, procurando realizar las validaciones que sean necesarias (longitudes, tipos numéricos,

tipos de fecha, etc.) lanzando una WebAccionExcepcion en caso de encontrar algún dato erróneo.

5.5.2 Generador de Eventos

El Evento es la clase que envuelve las peticiones al servidor. Contiene el nombre de la clase que debe procesar la acción, ligando por tanto la EJBAccion al evento, y los datos que deben pasársele a la EJBAccion.

La clase proporciona métodos para acceder a los campos, tanto en consulta como en modificación. También es posible cambiar dinámicamente el nombre de la clase que debe procesarlo, usando el método heredado *setNombreBeanAccion().*

5.5.2.1 Uso del generador

En principio estas clases, Eventos, siempre se generan usando el generador, ya que en ningún caso es necesario modificar el código generado. Los posibles cambios se limitan a modificar adecuadamente el fichero descriptor y generar el evento.

Como todos los descriptores de generación, este tiene formato XML. Su sintaxis es la siguiente:

```
<Evento  clase= "EventoDatos" accionEJB= "EJBAccionProcesarDatos" extends = "EventoBase">

        <campo nombre    = "campo1"tipo = "STRING"  />

        <campo nombre    = "campo2"tipo = "BOOLEAN"/>

        <campo nombre    = "campo3"tipo = "INT"/>

        <campo nombre    = "campo4"tipo = "LONG"    />

        <campo nombre    = "campo5"tipo = "FLOAT"   />

        <campo nombre    = "campo6"tipo = "LIST"/>

</Evento>
```

Ilustración 49 Ejemplo de descriptor XML para la generación de Evento

El nombre de la clase del Evento es obligatorio, así como el nombre de la EJBAccion. El atributo extends sirve para hacer que el evento generado herede de un evento ya existente. De esa forma es posible simplificar eventos que tengan campos en común. El evento del que se herede debe estar situado en el package tid.aplicacion.evento.

A continuación se debe definir una entidad campo por cada campo que se desee insertar en el evento. Los atributos necesarios para generar un campo son el nombre y el tipo. Existen una serie de tipos predefinidos, que son los mostrados en el ejemplo superior, que se corresponden con los tipos Java de la clase generada.

La clase del generador de Eventos es ***tid.puzzle.herramienta.generador.GeneradorEvento***, y está contenido en el jar dado.

Los parámetros que se le pasan al generador son los siguientes:

- **-f** : fichero xml de definición del Evento
- **-d** : Directorio destino donde se generará el código
- **-p** : Nombre del proyecto para el que se generarán las clases (influye en el paquete destino)
- **-h** : muestra la ayuda del generador.

Como en el apartado anterior, asumiendo que la librería del generador está en una carpeta generación, dentro de la carpeta raíz del proyecto, y que el proyecto se llama aplicación, una línea de comandos válida para invocar al generador podría ser la siguiente:

```
java -cp

generacion/generador-0.9.jar

tid.puzzle.herramienta.generador.GeneradorEvento -f generacion/generacionEvento.xml -d . -p aplicacion
```

Ilustración 50 Ejemplo de llamada para la generación de WebAccion

El generador crea la clase en el package *tid.aplicacion.evento*. A partir del atributo accionEJB, genera el nombre completo de la EJBAccion, considerando que está situada en el package *tid.aplicacion.controlador.ejb.accion*. Por ello las EJBAccion deben ir siempre creadas en ese directorio (recuérdese que las fuentes deben estar en el directorio fuentes, que debe colgar del directorio raíz del proyecto)

5.5.3 Generador de Acciones de Negocio: EJBAccion

Las EJBAccion son clases que procesan todas las acciones contra el modelo. Su misión es recibir un evento, e invocar a las clases de negocio necesarias para procesar la petición, devolviendo al FrameWork de Puzzle un EventoRespuesta con el resultado de la acción contra el modelo.

Las EJBAccion heredan de *tid.puzzle.framework.controlador.ejb.accion.AccionBase,* que proporciona una implementación vacía de los métodos más importantes. Los métodos *iniciar()* y *finalizar()* no suelen ser sobrecargados, salvo que se pretenda hacer operaciones de carga o liberación de elementos usados dentro del método *procesar().*

Evidentemente, el núcleo del procesamiento está en el método procesar, en el que debe invocarse a las clases de lógica de negocio necesarias para ello.

Las clases de negocio pueden crearse de diversas maneras. Pueden ser JavaBeans o simples clases Java, creados en cada llamada a la lógica, o usarse EJB's , usando el patrón Facade para su invocación. La elección queda delegada a los requisitos de la aplicación o preferencias del programador. El único requisito exigido por Puzzle es que el método devuelva un EventoRespuesta, indicando el resultado de la acción.

5.5.3.1 Uso del generador

Al igual que con la WebAccion, el generador de Puzzle para la EJBaccion solo genera el esqueleto de la clase, cuyos métodos posteriormente es necesario dotar de contenido. La definición de la EJBAccion es análoga a la de la WebAccion

```
<AccionEJB clase="EJBAccionCargaDatos" />
```

Ilustración 51 Ejemplo de descriptor XML para la generación de EJBAccion

Donde el atributo clase indica el nombre de la EJBAccion a generar. La clase del generador de EJBaccion es *tid.puzzle.herramienta.generador.GeneradorEJBAccion*, y está contenido en el jar dado. Los parámetros que se le pasan al generador son los mismos que se han descrito en los dos generadores anteriores pero proporcionado el XML apropiado.

Igualmente, la invocación del generador es muy similar al resto de los generadores, como se refleja en la línea de comandos del generador:

```
java -cp

generacion/generador-0.9.jar

tid.puzzle.herramienta.generador.GeneradorEJBAccion -f generacion/generacionEJBAccion.xml -d . -p aplicacion
```

Ilustración 52 Ejemplo de llamada para la generación de EJBAccion

Las EJBAccion así generadas se crean siempre en el paquete *tid.aplicacion.controlador.ejb.accion*. La estructura que generan es la siguiente:

```
package tid.aplicacion.controlador.ejb.accion;

// Trazas

import org.apache.log4j.Logger;

// Puzzle

import tid.puzzle.framework.controlador.ejb.accion.AccionBase;

import tid.puzzle.framework.evento.Evento;

import tid.puzzle.framework.evento.EventoRespuesta;

import tid.puzzle.framework.evento.EventoExcepcion;

public class EJBAccionBorrarEntrada extends AccionBase

{

        static Logger _logger = Logger.getLogger(EJBAccionBorrarEntrada.class);
```

```
        public EventoRespuesta procesar (Evento ev) throws EventoExcepcion
        {
                // Aqui tu código
                return null;
        } // procesar
} // class
```

Ilustración 53 Resultado de la generación de EJBAccion

El generador proporciona un esqueleto que ya implementa todos los métodos necesarios. Solo es necesario, al igual que con la WebAccion, proporcionarles contenido, implementando las invocaciones a lógica de negocio que sean más adecuadas, ya sea usando JavaBeans o EJB.

5.5.4 Generador de Eventos de Respuesta

El EventoRespuesta es el elemento que se devuelve al FrameWork de PUZZLE indicando el resultado de la acción contra modelo. Al contrario que el Evento, no está ligado a una EJBAccion, y cualquiera EJBAccion puede devolver un EventoRespuesta.

La clase base del EventoRespuesta es *tid.puzzle.framework.evento.EventoRespuestaBase*, que proporciona los miembros y métodos necesarios para devolver un resultado.

Dichos métodos son *getResultado*, *setResultado*, *getComentario* y *setComentario*.

El campo resultado es un valor booleano que indica el resultado de la acción. El campo comentario es un *String* que puede usarse para devolver comentarios o detalles desde la lógica de negocio a la capa del controlador, y a través de ella a la capa de vista.

5.5.4.1 Uso del generador

Estas clases, al igual que los Eventos, siempre se generan usando el generador, ya que en ningún caso es necesario modificar el código generado. Los posibles cambios se limitan a modificar adecuadamente el fichero descriptor y generar el evento. Como todos los descriptores de generación, este tiene formato XML. Su sintaxis es la siguiente:

```
<EventoRespuesta clase= "EventoRespuestaAnotacion" extends = "EventoRespuestaBase">
        <campo nombre    = "campo1"tipo = "STRING" />
        <campo nombre    = "campo2"tipo = "BOOLEAN"/>
```

```
        <campo nombre    = "campo3"tipo = "INT"/>
        <campo nombre    = "campo4"tipo = "LONG"    />
        <campo nombre    = "campo5"tipo = "FLOAT"   />
        <campo nombre    = "campo6"tipo = "LIST"/>
</EventoRespuesta>
```

Ilustración 54 Ejemplo de descriptor XML para la generación de EventoRespuesta

Al igual que en la generación del Evento, el nombre de la clase es obligatorio, pero en este caso la entidad EventoRespuesta no tiene un atributo accionEJB, porque no están asociadas a un clase de EJBAccion. Si es posible hacer que extiende de un EventoRespuesta ya definido, que debe estar dentro del package *tid.aplicacion.evento.*

El EventoRespuesta también puede contener campos, que se definen exactamente igual que en el Evento, siempre recordando que la herencia ya proporciona los métodos necesarios para contener la información de resultado y comentario.

Finalmente, el generador del EventoRespuesta es el mismo que el usado para generar los Eventos, y su forma de invocación es la misma, por lo que no se repetirá aquí el proceso de generación.

5.5.5 Generador de Manejadores de Flujo

El ManejadorFlujo es el elemento que redirige el flujo de la acción a una pantalla u otra para devolver al usuario. Su existencia no es imprescindible, y en caso de que una acción no tenga definida un ManejadorFlujo, el flujo se redirige a la pantalla destino definida en la propia entidad del Mapeo.

El ManejadorFlujo no está ligado a la existencia de lógica de negocio asociada a una acción. De hecho, es perfectamente posible definir un Mapeo que no contenga claseAccion (WebAccion), pero si tenga ManejadorFlujo. Lo contrario es mucho más raro, pues lo habitual es usar el ManejadorFlujo para, en función del resultado devuelto por la lógica de negocio, redirigir a una pantalla u otra.

Al igual que las demás clases activas del ciclo de vida de la acción (WebAccion y EJBAccion) la clase ManejadorFlujo debe implementar los métodos *iniciar(), procesar() y finalizar(),* aunque la implementación base proporcionada por Puzzle ya provee de una implementación vacía de dichos métodos, por lo que en la mayoría de los casos solo será necesario sobrecargar el método de *procesar().*

El método *procesar()* debe implementar la lógica necesaria para decidir la pantalla que debe presentarse al usuario, devolviendo un String que debe corresponderse con una de las entidades Resultado definidas en el fichero mapeos.xml. De esa forma Puzzle sabrá, una vez realizado el procesamiento de la acción, que pantalla de destino debe devolver al usuario.

Anteriormente se ha comentado que el ManejadorFlujo no es imprescindible. Sin embargo, en muchas ocasiones será necesario cargar datos de diversos orígenes (a través de los DAOs, por ejemplo), para generar la pantalla que se devuelve al usuario.

El mecanismo habitual es realizar esa carga en el ManejadorFlujo, en su método *procesar(),* y los resultados de las consultar se carguen como atributos en la request, -usando constantes definidas en la clase constantes -, para que estén disponibles en las JSP que componen las pantallas que generarán la pantalla a mostrar al usuario.

Un ejemplo típico del ManejadorFlujo podría ser el siguiente:

```
package tid.aplicacion.controlador.web.flujo;


// Jdk

import javax.servlet.http.HttpServletRequest;


// Trazas

import org.apache.log4j.Logger;


// Puzzle

import tid.puzzle.framework.controlador.web.flujo.ManejadorFlujo;

import tid.puzzle.framework.controlador.web.flujo.ManejadorFlujoBase;

import tid.puzzle.framework.controlador.web.flujo.ManejadorFlujoExcepcion;

import tid.puzzle.framework.evento.EventoRespuesta;


// Puzzle - Validacion

import tid.puzzle.framework.util.Validador;


//Aplicacion

import tid.aplicacion.util.Constantes;

public class ManejadorFlujoCargaDatos extends ManejadorFlujoBase

{

	static Logger _logger = Logger.getLogger(ManejadorFlujoCargaDatos.class);

	public String procesarFlujo (HttpServletRequest request) throws ManejadorFlujoExcepcion

	{

		EventoRespuesta evRespuesta = null;

		evRespuesta=(EventoRespuesta)request.getAttribute(Constantes.ATRB_EVENTO_RESPUESTA);

		String pantallaDestino=null;

		if (evRespuesta.getResultado())

		{

			_logger.debug("Acción realizada con exito");
```

```
                    pantallaDestino="correcto"
                    //Carga de datos
                    .....
                    //Fin de la carga de datos
                }
                else
                {
                    _logger.debug("La acción falló");
                    pantallaDestino="error"
                    //Carga de datos
                    .....
                    //Fin de la carga de datos
                }
                return pantallaDestino;
        } // procesar
        public void finalizar (HttpServletRequest request)
        {
                // Aqui tu código
        }
} // class
```

Ilustración 55 Ejemplo de fuente ManejadorFlujo

En este ejemplo, se recoge de la request el EventoRespuesta que ha dejador la WebAccion en la request. Obsérvese la manera de pasar los resultados desde la capa de negocio a la capa de presentación.

5.5.5.1 Uso del generador

El generador de Puzzle para el ManejadorFlujo solo genera el esqueleto de la clase, cuyos métodos posteriormente es necesario dotar de contenido. La definición del ManejadorFlujo es análoga a la de la WebAccion y la EJBAccion:

```
<ManejadroFlujo clase="ManejadorFlujoCargaDatos" />
```

Ilustración 56 Ejemplo de descriptor XML para la generación de ManejadorFlujo

Donde el atributo clase indica el nombre del ManejadorFlujo a generar. La clase del generador de ManejadorFlujo es ***tid.puzzle.herramienta.generador.GeneradorManejadorFlujo***, y está contenido en el jar dado. La invocación y los parámetros son los mismos descritos en los generadores anteriores.

Los ManejadorFlujo generados se crean siempre en el paquete *tid.aplicacion.controlador.web.flujo*. La estructura que generan es la siguiente:

```
package tid.aplicacion.controlador.web.flujo;

// Jdk

import javax.servlet.http.HttpServletRequest;

// Trazas

import org.apache.log4j.Logger;

// Puzzle

import tid.puzzle.framework.controlador.web.flujo.ManejadorFlujo;

import tid.puzzle.framework.controlador.web.flujo.ManejadorFlujoBase;

import tid.puzzle.framework.controlador.web.flujo.ManejadorFlujoExcepcion;

// Puzzle - Validacion

import tid.puzzle.framework.util.Validador;

public class ManejadorFlujoCargaDatos extends ManejadorFlujoBase

{

	static Logger _logger = Logger.getLogger(ManejadorFlujoCargaDatos.class);

	public void iniciar(HttpServletRequest request)

	{

		// Aqui tu código de inicialización

	}

	public String procesarFlujo (HttpServletRequest request) throws ManejadorFlujoExcepcion

	{

		// Aqui tu código

		return null;

	} // procesar

	public void finalizar (HttpServletRequest request)

	{

		// Aqui tu código

	}
```

```
} // class
```

Ilustración 57 Ejemplo de fuente generado ManejadorFlujo

El generador proporciona un esqueleto que ya implementa todos los métodos necesarios. De hecho, puesto que la clase *ManejadorFlujoBase* ya proporciona una implementación de los tres métodos, si no se usa el *iniciar* o el *finalizar*, no es necesario crearlos. Una consideración importante es que si en el ManejadorFlujo se usan objetos que sea preciso liberar (como los DAOs) el lugar indicado para ello es el método *finalizar*.

5.6 Componentes adicionales: Facilidades

A continuación, se describen diversos componentes adicionales de Puzzle, que no entran en el ciclo de trabajo de Puzzle propiamente dicho, sino que aportan funcionalidad a la aplicación. Estos son: el componente DAO (Acceso a datos), el módulo de seguridad, gestor transaccional, etc.

5.6.1 Generador DAO

Puzzle Framework cuenta con un modelo de acceso a datos basado en DAOs que soporta múltiples orígenes de datos, concretamente BD, LDAP y XML. El objetivo es dotar a la información persistente de una representación como objeto.

El modelo de DAOs de Puzzle se basa en el patrón DAO propuesto en los patrones de diseño J2EE [23], con la particularidad de que el modelo implementado impone una relación 1-1 entre el DAO de acceso a los datos y la unidad de almacenamiento en la fuente de datos (tabla en BD, nodo en LDAP o fichero XML).

La capa de acceso a datos permite la consulta, inserción, modificación y borrado de entidades en las fuentes persistentes de datos.

Además, el DAO ofrece una envoltura EJB para su uso en entornos donde el uso de EJB facilite el trabajo. Por otro lado, el DAO proporciona también un modelo basado en el patrón FastLane, que permite un rápido acceso a los datos.

Actualmente los DAOs sobre BD y LDAP ofrecen la posibilidad de usar transacciones, posibilidad que no se ofrece en el DAO de XML. En el siguiente diagrama de clases se ilustra el modelo de componentes que proporcionan la infraestructura DAO descrita:

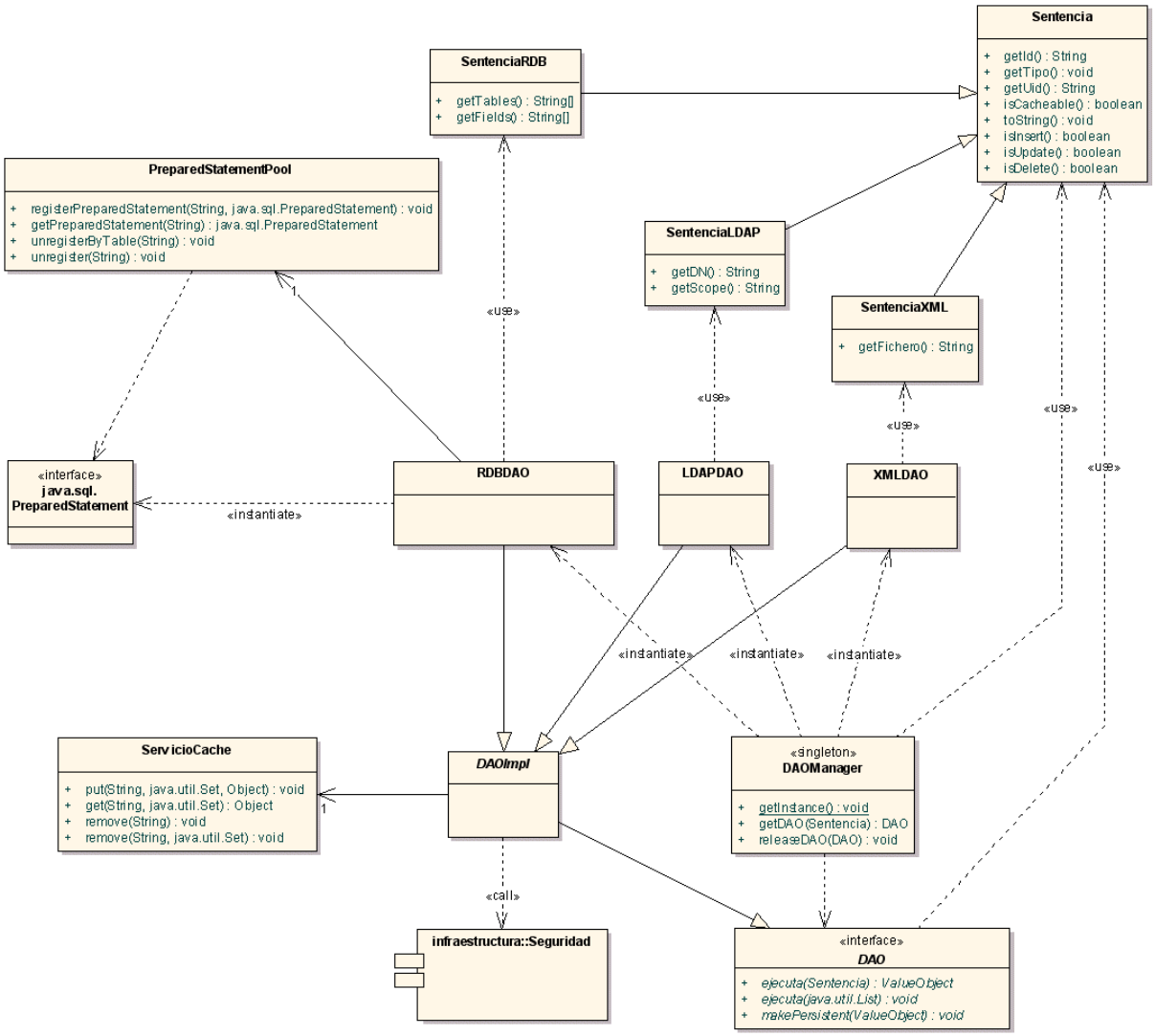

Ilustración 58 Diagrama de Clases del DAO de Puzzle

Y el siguiente diagrama de secuencia ilustra un escenario de uso del DAO de Puzzle:

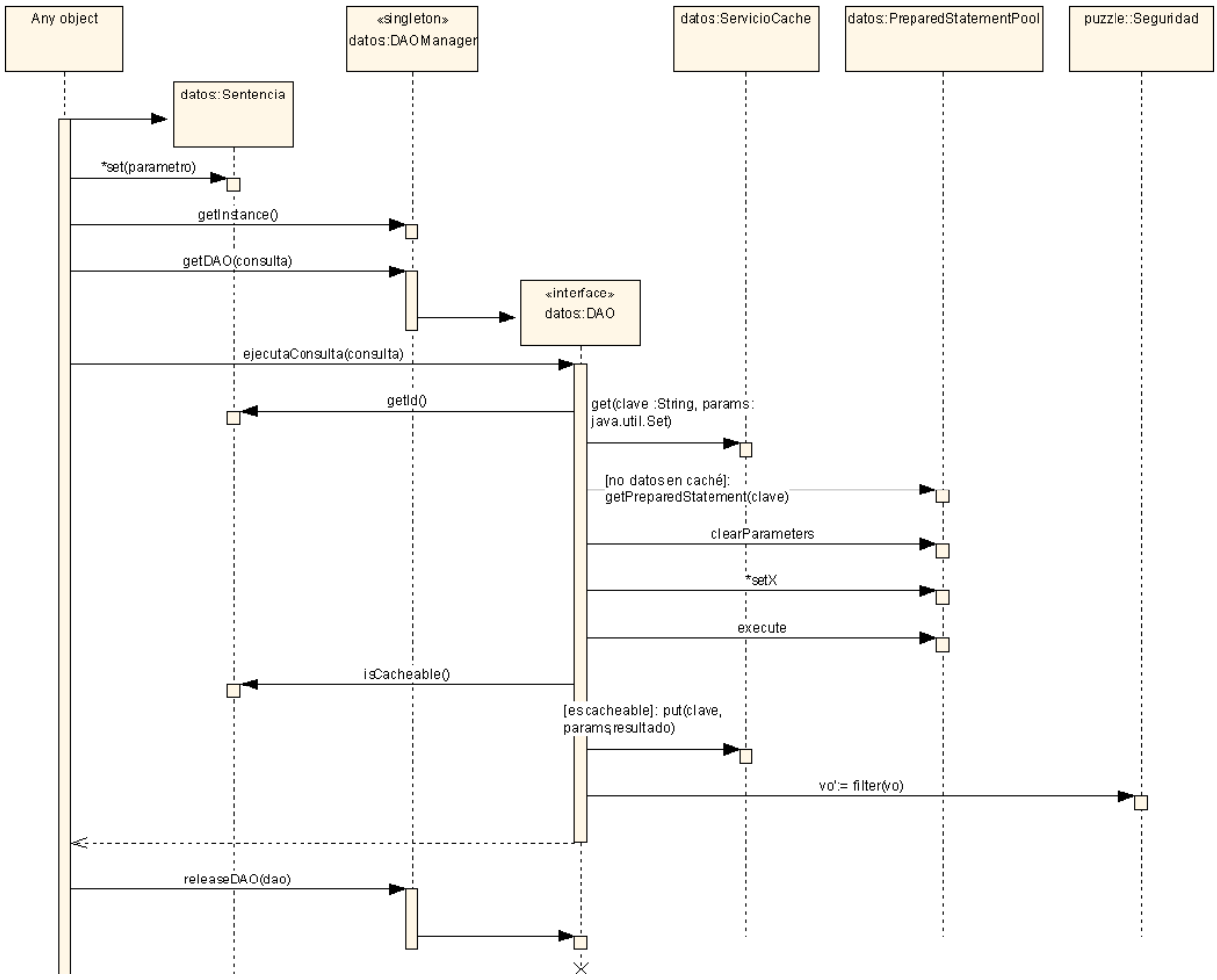

Ilustración 59 Diagrama de Secuencia del DAO de Puzzle

5.6.1.1 Uso del generador

El DAO es el componente de Puzzle más potente y versátil que puede ser construido usando generación automática de código. Al igual que todos los componentes generados de Puzzle, éste se debe describir en un fichero XML.

A la hora de definir el DAO, primero se debe diferenciar el tipo de DAO que se va a usar, a elegir entre el DAO para BD, para LDAP y para XML. Sin embargo, todo DAO presenta unos elementos comunes, tal y como se describen a continuación:

- **ValueObject**: Todos los DAOs deben definir un ValueObject base, que es el objeto que representa en Puzzle a un dato persistente en la fuente de datos. En el caso de los DAOs, el ValueObject se corresponde con un registro de la tabla sobre la que se construye el DAO. En el caso de un DAO XML, se corresponde con una entidad principal del XML, y en el caso del LDAP, se corresponde con un nodo.

  Todo ValueObject base contiene, por definición, dos campos: ***ENTRYID*** y ***TIME_STAMP***. El primero de ellos es un campo de tipo int, que no puede ser nulo, y es único en el origen de datos (tabla, entidad o nodo). El segundo de ellos es un campo de tipo long, que contiene el sello de tiempo de la inserción o última modificación del elemento en la fuente persistente. Estos dos campos

son obligatorios, y por corresponderse con el elemento en la fuente de datos persistente, éste debe tener igualmente estos campos.

Adicionalmente, en Puzzle se suele usar un tercer campo *NO_USER*, campo de tipo String de 20 caracteres, que se usa para anotar en cada registro el último usuario que lo insertó/modificó.

El ValueObject base así definido es el usado en las operaciones de inserción, modificación y borrado, pero no en las de consulta, donde es posible definir ValueObject alternativos que reflejen las joins o consultas relacionadas sobre la fuente de datos.

- **Consultas**: Dentro de un DAO se pueden definir consultas, que implementará el FastLane asociado al DAO en cuestión. Dichas consultas se componen de un conjunto de restricciones sobre los datos (la condición de consulta propiamente dicha), de un objeto de entrada, usado para pasar parámetros a la condición de consulta, y de un objeto de salida (ValueObject) que puede ser único, o ser una lista de objetos.

- **EJB y FastLane**: Nombres de las clases del DAO que proporcionan la envoltura EJB del DAO y del FastLane de consultas del DAO respectivamente.

Un ejemplo de descriptor de DAO, para el caso del tipo relacional, es el mostrado en la siguiente figura:

```
<dao modulo="moduloDAO">
        <ValueObject clase="NombreVO" dataSource="nombrePool">
                <mapeos tabla="NOMBRE_TABLA"/>
                <sequencia nombre="NOMBRE_SECUENCIA"/>
                <campo nombreVO="nombreCampo1" tipoJ="INT">
                        <sql nombreDB="NOMBRE_CAMPO_1" tipoDB="NUMBER"/>
                </campo>
                <campo nombreVO="nombreCampo2" tipoJ="STRING">
                        <sql nombreDB="NOMBRE_CAMPO_2" tipoDB="VARCHAR2"/>
                </campo>
                <campo nombreVO="nombreCampo3" tipoJ="BOOLEAN">
                        <sql nombreDB="NOMBRE_CAMPO_3" tipoDB="VARCHAR2"/>
                </campo>
                <campo nombreVO="nombreCampo4" tipoJ="DATE">
                        <sql nombreDB="NOMBRE_CAMPO_4" tipoDB="DATE"/>
```

```
            </campo>
            <campo nombreVO="noUser" tipoJ="STRING">
                <sql nombreDB="NO_USER" tipoDB="VARCHAR2"/>
            </campo>
        </ValueObject>
        <Consulta nombreMetodo="consulta1" sql="
            SELECT ENTRYID, TIME_STAMP,
                NOMBRE_CAMPO_1,
                NOMBRE_CAMPO_2,
                NOMBRE_CAMPO_3,
                NOMBRE_CAMPO_4,
                NO_USER
             FROM NOMBRE_TABLA
            WHERE NOMBRE_CAMPO_1 = ? ">
            <Entrada clase="CstEntrada">
                <campo nombreVO="nombreCampo1"   tipoJ="INT"  tipoDB="NUMBER"/>
            </Entrada>
            <Salida esLista="false" esVOBasico="true" clase="NombreVO"/>
        </Consulta>
        <Consulta nombreMetodo="consulta2" sql="
            SELECT NOMBRE_CAMPO_1,
                NOMBRE_CAMPO_2,
                NOMBRE_CAMPO_3,
                NO_USER
             FROM NOMBRE_TABLA
            WHERE NOMBRE_CAMPO_1 = ? ">
            <Entrada clase="CstEntrada">
                <campo nombreVO="nombreCampo1"   tipoJ="INT"  tipoDB="NUMBER"/>
            </Entrada>
            <Salida esLista="true" esVOBasico="false" clase="NombreVOAlternativo">
                <campo nombreVO="nombreCampo1"  tipoJ="INT"    tipoDB="NUMBER"  />
                <campo nombreVO="nombreCampo2" tipoJ="STRING" tipoDB="VARCHAR2"/>
                <campo nombreVO="nombreCampo3" tipoJ="BOOLEAN" tipoDB="VARCHAR2"/>
                <campo nombreVO="noUser"      tipoJ="STRING"  tipoDB="VARCHAR2"/>
```

```
            </Salida>
        </Consulta>
        <Consulta nombreMetodo="consultaDinamica" sql="
                SELECT NOMBRE_CAMPO_1,
                    NO_USER
                FROM NOMBRE_TABLA
                WHERE">
            <Entrada clase="CstDinamica">
            </Entrada>
            <Salida esLista="true" esVOBasico="false" clase="NombreVOAlternativo">
                <campo nombreVO="nombreCampo1"  tipoJ="INT"    tipoDB="NUMBER"  />
                <campo nombreVO="noUser"      tipoJ="STRING"  tipoDB="VARCHAR2"/>
            </Salida>
        </Consulta>
        <Ejb nombre="EJBNombreVO"/>
        <FastLane nombre="NombreVOFastLane"/>
</dao>
```

Ilustración 60 Ejemplo de descriptor XML para la generación de DAO

Donde:

- **dao**: Entidad básica de descripción del DAO.
  - **modulo**: define el package donde se creará el DAO (*tid.proyecto.modulo*)
  - **ValueObject**: Entidad básica de definición del ValueObject base del DAO. Debe corresponder campo a campo con la tabla de BD sobre la que se operé(Exceptuando los campos ENTRYID y TIME_STAMP, que son generados automáticamente)
    - **Clase**: nombre del ValueObject.
    - **DataSource**: Nombre del datasource sobre el que operará el DAO (Véase 5.4.1 Descriptor de la aplicación web)
    - **Mapeos**: Entidad que describe la relación entre tabla y ValueObject.
      - **Tabla**: Nombre de la tabla que se corresponde con el ValueObject.
    - **Sequencia**: Entidad que define la secuencia de la que se extraerán los entryid.
      - **Nombre**: Nombre de la secuencia de laque se extraerán los entryid.

        - **Campo**: Entidad que define un campo en el ValueObject y su relación con un campo de la BD.
            - **NombreVO**: Nombre del miembro en el ValueObject
            - **TipoJ**: Tipo del miembro en el ValueObject.
        - **Sql**: Entidad que define el campo en BD
            - **NombreDB**: Nombre del campo en BD
            - **tipoDB**: Tipo del campo en BD

    - **Consulta**: Entidad que define una consulta.
        - **nombreMetodo**: Nombre del método de consulta en el FastLane.
        - **sql**: Sentencia sql que define la consulta. Los campos deben corresponderse en orden con los del ValueObject de salida, tanto si es un ValueObject base como si se define a medida. En caso de ser un ValueObject base, debe tenerse en cuenta que el primer y segundo campo de la select deben ser siempre ENTRYID y TIME_STAMP respectivamente.
        - **Entrada**: Entidad que define el objeto de entrada a la consulta para pasarle parámetro. Puede ser void.
            - **Clase**: Nombre de la clase que se le pasa al FastLane con los parámetros de búsqueda. Si se declara void, no es necesario definir el resto del objeto.
        - **Campo**: Entidad que define un parámetro de la consulta
            - **NombreVO**: nombre del miembro en el objeto Entrada.
            - **TipoJ**: Tipo del miembro en el objeto Entrada.
            - **TipoDB**: Tipo del campo correspondiente en BD.
        - **Salida**: Entidad que define el ValueObject de salida de la consulta.
            - **EsLista**: valor booleano que indica si la consulta devolverá un solo ValueObject o una lista de ValueObjects
            - **EsVOBasico**: valor booleano que indica si el ValueObject de salida es el ValueObject base.
            - **Clase**: Nombre de la clase del ValueObject de salida.
            - **Campo**: Si el ValueObject de salida no es básico, es necesario definir sus campos.
                - **NombreVO**: nombre del miembro en el ValueObject de salida.
                - **TipoJ**: Tipo del miembro en el ValueObject de salida.
                - **TipoDB**: Tipo del campo correspondiente en BD.

    - **Ejb**: Nombre de la clase EJB que envolverá al DAO.
    - **FastLane**: Nombre de la clase de consultas.

A partir del descriptor se puede generar DAOs con todas las consultas que sean necesarias, usando diversos objetos de entrada y ValueObjects de salida.

Puzzle también admite el uso de consultas dinámicas, en las que solo se establece los campos a mostrar y las tablas de partida. Para definirlas, hay que crear una consulta con toda la query, pero que acabe en la cláusula *WHERE*, sin indicar condiciones, y crear un objeto de entrada vacio. En apartados posteriores se explicará el uso de estas consultas.

5.6.1.2 Uso del DAO

El componente DAO generado por Puzzle comprende múltiples clases, donde la estructura de carpetas o paquetes puede verse en la siguiente figura:

Ilustración 61 Estructura de paquetes de un DAO

Donde:

- La carpeta ***consulta*** contiene las clases usadas como objetos de entrada de datos para las consultas que proporciona el DAO generado.
- La carpeta ***dao*** contiene la clase del DAO propiamente dicho, entre otros elementos, de los cuales el más importante es la factoría, que permite abstraer la instanciación del DAO.
- La carpeta ***ejb*** contiene la envoltura EJB del DAO. Recuérdese que si se usa esta envoltura, es necesario declarar la EJB en *ejb-jar.xml.*
- La carpeta ***fastlane*** contiene el FastLane para facilitar las consultas.
- La carpeta ***valueobject*** contiene las clases que usadas por el DAO para devolver los resultados de las consultas, y el ValueObject base usado para las inserciones, actualizaciones y borrados de elementos, es decir, contiene la información del negocio propiamente dicha que se almacena persistentemente.

Las posibles acciones que se pueden realizar con el DAO son las siguientes:

- **Consulta**: Las consultas definidas en el descriptor del DAO se usan de la siguiente manera:

```
DocumentoFastLane docFastLane = null;
CstDocumentosPorTipoActYSistema inCst = null;
List listaDocumentos = null;
try
{
        //Se instancian los elementos necesarios para la consulta
        docFastLane = new DocumentoFastLane();
        inCst = new CstDocumentosPorTipoActYSistema();
        //Se cargan los parámetros de la consulta
        inCst.setTactCoTipoActuacion(tipoActuacion);
        inCst.setSistCoSistema(codigoSistema);
        //Se ejecuta la consulta
        listaDocumentos = docFastLane.consultaDocumentosPorTipoActYSistema(inCst).getLista();
}
catch (DAOFinderException dfe)
{
        _logger.debug("No existen documentos”);
}
finally
{
        //Se liberan los elementos
        if (inCst!=null)
                inCst.libera();
        inCst=null;

        if (docFastLane!=null)
                docFastLane.libera();
        docFastLane =null;
}
```

Ilustración 62 Ejemplo de uso del DAO: Consultas

Si al definir la consulta en el descriptor del DAO se puso a *true* el indicador de resultado múltiple, el FastLane devuelve un objeto *DatosConsulta*. Ese objeto tiene un método *getLista()* que devuelve una *java.util.List* conteniendo los ValueObjects definidos como salida en la consulta. En caso contrario, el FastLane devuelve directamente el ValueObject.

- **Inserción**: Cuando se explicaron los elementos comunes del DAO, se dijo que los ValueObject base tienen un *ENTRYID* que es necesario rellenar con un valor único en la fuente de datos persistente. El DAO proporciona un método que devuelve un entryID nuevo a partir de la secuencia definida en el descriptor del DAO (secuencia que debe existir en base de datos). La manera habitual de trabajar es implementar en la clase que use el DAO, unos métodos que abstraigan el proceso de obtención del DAO.

```
protected Documento insertaDocumento(Documento documento) throws Exception
{
	DocumentoDAO dao=	null;
	try
	{
		dao = this.getDocumentoDAO();
		dao.insertar(documento);
	}
	finally
	{
		if(dao != null)
			dao.libera();
		dao = null;
	}
	return documento;
}
protected int dameEntryDocumento()throws Exception
{
	DocumentoDAO dao= null;
	int entryid = -1;
```

```
		try
		{
			dao = this.getDocumentoDAO();
			entryid=dao.dameEntry();
		}
		finally
		{
			if(dao != null)
				dao.libera();
			dao = null;
		}
		return entryid;
}
private DocumentoDAO getDocumentoDAO() throws Exception
{
		return DocumentoDAOFactory.dameDAO();
}
```

Ilustración 63 Ejemplo de uso del DAO: Inserción (1)

De esta forma, el proceso de obtención de inserción se limita a lo siguiente:

```
Documento documento =null;
try
{
		documento = new Documento();
		// Se cargan los campos del ValueObject base
		...
		// Una vez cargado se solicita un entryid
		documento.setEntryid(dameEntryDocumento());
		// Se inserta
		insertaDocumento(documento);
}
catch(Exception ex)
```

```
{
        _logger.error("Se produjo un error al insertar : ",ex);
}
finally
{
        if(documento!= null)
                documento.libera();
        documento = null;
}
```

Ilustración 64 Ejemplo de uso del DAO: Inserción (2)

A menudo, los ValueObjects se definen con un campo de Código que debe ser único; un campo distinto del ENTRYID, para uso de la aplicación. En ese caso, lo más sencillo es generar dicho código usando el propio entryID

```
...
// Una vez cargado se solicita un entryid
documento.setEntryid(dameEntryDocumento());
documento.setDocuCoDocumento(documento.getEntryid());
// Se inserta
insertaDocumento(documento);
...
```

Ilustración 65 Ejemplo de uso del DAO: Inserción (3)

De esta forma ambos códigos son únicos, y se evitan duplicidades. El campo *TIME_STAMP* se rellena automáticamente en la inserción.

- **Actualización**: A la hora de actualizar un elemento, lo habitual es realizar la consulta que se desee en el origen de datos, para obtener el elemento a modificar. Posteriormente, se actualizan sus campos, y se procede a la actualización. El *ENTRYID* no debe modificarse, y el campo *TIME_STAMP* debe ser actualizado utilizando, en principio, el *TIME_STAMP* del sistema. Para obtener este valor se puede recurrir al *ControladorClienteWeb* (desde la WebAccion) o al *ControladorCliente* (desde la EJBAccion), ya que ambos poseen un método *getTimeStamp()* que devuelve el *TIME_STAMP* de la última consulta. Ese Timestamp debe pasarse a la clase de lógica de negocio que realice la actualización.

Al igual que en el caso de la inserción lo habitual es usar un método que abstraiga la invocación del DAO.

```
protected Documento updateDocumento(Documento documento) throws Exception
{
	DocumentoDAO dao= null;
	try
	{
		dao = this.getDocumentoDAO();
		dao.update(documento);
	}
	finally
	{
		if(dao != null)
			dao.libera();
		dao = null;
	}
	return documento;
}
```

Ilustración 66 Ejemplo de uso del DAO: Actualización (1)

Evidentemente en este caso no hay que solicitar un nuevo *ENTRYID* porque se obtiene a partir de la consulta (la entrada ya existe y tiene asignado un entryid)

```
DocumentoFastLane docFastLane = null;
CstDocumentoPorCodigo inCst = null;
Documento documento = null;
try
{
	//Se instancian los elementos necesarios para la consulta
	docFastLane = new DocumentoFastLane();
	inCst = new CstDocumentoPorCodigo ();
	//Se cargan los parámetros de la consulta
	inCst. docuCoDocumento(codigoDocumento);
	//Se ejecuta la consulta
```

```
            documento = docFastLane.consultaDocumentoPorCodigo(inCst);
            // Se actualizan los campos del ValueObject base
            ...
            // Una vez actualizado, se carga el nuevo timeStamp
            documento.setTimeStamp(timeStamp);
            // Se actualiza
            updateDocumento (documento);
}
catch (DAOFinderException dfe)
{
            _logger.debug("No existen documentos”);
}
catch (DAODBUpdateException due)
{
            _logger.debug("Hubo un error al actualizar el documento”,due);
}
finally
{
            //Se liberan los elementos
            if (inCst!=null)
                        inCst.libera();
            inCst=null;
            if (docFastLane!=null)
                        docFastLane.libera();
            docFastLane =null;
            if (documento!=null)
                        documento.libera();
            documento =null;
}
```

Ilustración 67 Ejemplo de uso del DAO: Actualización (2)

- **Borrado**: El borrado de elementos usando el DAO de Puzzle es muy sencillo, pues basta con pasarle al DAO un ValueObject base cuyo *ENTRYID* sea el del elemento a borrar. Por lo tanto sería:

```
protected void borraDocumento(Documento documento) throws Exception
{
	DocumentoDAO dao= null;
	try
	{
		dao = this.getDocumentoDAO();
		dao.borrar(documento);
	}
	finally
	{
		if(dao != null)
			dao.libera();
		dao = null;
	}
}
.....
DocumentoFastLane docFastLane = null;
CstDocumentoPorCodigo inCst = null;
Documento documento = null;
try
{
	//Se instancian los elementos necesarios para la consulta
	docFastLane = new DocumentoFastLane();
	inCst = new CstDocumentoPorCodigo ();
	//Se cargan los parámetros de la consulta
	inCst. docuCoDocumento(codigoDocumento);
	//Se ejecuta la consulta
	documento = docFastLane.consultaDocumentoPorCodigo(inCst);
	// Se borra
	borraDocumento (documento);
}
catch (DAOFinderException dfe)
{
	_logger.debug("No existen documentos”);
```

```
}
catch (Exception ex)
{
        _logger.debug("Hubo un error al borrar el documento",ex);
}
finally
{
        //Se liberan los elementos
        if (inCst!=null)
                inCst.libera();
        inCst=null;
        if (docFastLane!=null)
                docFastLane.libera();
        docFastLane =null;
        if (documento!=null)
                documento.libera();
        documento =null;
}
```

Ilustración 68 Ejemplo de uso del DAO: Borrado

5.6.2 Gestor Transaccional

Puzzle Framework ofrece una funcionalidad de control de transacciones propia, basada en JTA, para controlar desde el propio código de la aplicación el principio y fin de las transacciones. Para su uso, es necesario que el contenedor J2EE ofrezca una implementación de JTA, o bien proporcionarla mediante una librería externa, como por ejemplo, JOTM [24].

El concepto de transacciones es aplicable a DataSource, por lo que aquellos orígenes de datos basados en DataSource (Base de datos y LDAP) pueden hacer uso de las transacciones. El único requisito es que el driver del DataSouce implemente el protocolo XA para transacciones distribuidas (más de un origen de datos involucrado).

El uso del GestorTransaccional es muy sencillo, pues se reduce a instanciar al Gestor, invocar el inicio de la transacción, ejecutar la lógica de negocio, y en caso de que el resultado sea correcto, aceptarla, cancelándola en caso contrario.

```
[…]
GestorTransaccional transaccion = null;
```

```
try
{
	// 1. - Se crea un gestor transaccional
	transaccion = new GestorTransaccional();
	// 2. - se inicia la Transacción.
	transaccion.iniciarTransaccion();
	_logger.debug("Transaccion: " + transaccion.getStrEstado());
	// 3. - Se procesa la lógica de negocio
	respuesta = this.procesaLogicaNegocio (ev);
	if (respuesta==true)
	{
		// 4. - Si todo ha ido bien aceptamos los cambios
		transaccion.aceptarTransaccion();
		_logger.debug("Transaccion (Commit): Realizado");
	}
	else
	{
		// 5. - Si algo ha ido mal deshacemos los cambios
		_logger.error("Iniciando Rollback");
		transaccion.cancelarTransaccion();
		_logger.debug("Transaccion: " + transaccion.getStrEstado());
	}
}
catch (LocalizadorRecursosExcepcion e)
{
	_logger.error("ERROR GENERAL DEL SISTEMA: no se localiza el gestor de transacciones ", e);
}
catch (GestorTransaccionalExcepcion e)
{
	_logger.error("ERROR GENERAL DEL SISTEMA: problema con las transacciones ", e);
}
catch (Exception e)
{
	_logger.error("Excepción inesperada: " + e.getMessage(), e);
```

```
}
finally
{
        // 6. - Se libera la transaccion
        if (transaccion != null)
                transaccion.libera();
        transaccion = null;
}
[...]
```

Ilustración 69 Ejemplo de fuente de uso del GestorTransaccional

5.6.3 Trazas

El sistema de trazas en el que se apoya Puzzle Framework se basa en el software libre **Log4j** [25] altamente reconocido por su rendimiento y funcionalidad.

Aún así, se realiza un modelo de recubrimiento mediante interfaces que permite utilizar cualquier otro sistema diferente siempre y cuando se cumplan las interfaces citadas.

En este caso, las trazas se configuran en el fichero *log4j.properties*, que debe estar localizado en la carpeta ***/WEB-INF/*** de la aplicación web. Su sintaxis puede consultarse en la propia documentación del framework Log4j usado. En los ejemplos ilustrados anteriormente, podemos ver el uso del mismo.

5.6.4 Librería de etiquetas (Tags)

Puzzle proporciona una librería de etiquetas (JSP Tags Library) para agilizar y homogenizar la creación de las pantallas de nuestra aplicación web otorgando un valor añadido en cuanto a la gestión dinámica de las pantallas.

Los tags de Puzzle no se encuentran dentro del alcance del documento aunque su utilización e incorporación a nuestros proyectos es muy sencilla, básicamente se realiza importando en nuestras JSPs el TLD correspondiente a la librería de Puzzle, como en general, para cualquier librería de Tags, tal y como se ilustra en la siguiente figura:

```
<%@ taglib uri="/WEB-INF/puzzle.tld" prefix="puzzle" %>
```

Ilustración 70 Ejemplo de uso de la librería de Tags Puzzle (página JSP)

Opcionalmente, para que dicha librería sea accesible para nuestra aplicación, hay que definirla en el fichero de despliegue de la aplicación (web.xml).

```
<taglib>
        <taglib-uri>/WEB-INF/puzzle.tld</taglib-uri>
        <taglib-location>/WEB-INF/puzzle.tld</taglib-location>
</taglib>
```

Ilustración 71 Ejemplo de descriptor web para el uso de la librería de Tags Puzzle (web.xml)

Podemos observar que, en cualquier caso, el fichero con la descripción de la librería debe estar localizado en la carpeta ***/WEB-INF/*** de la aplicación web.

6 Manual de usuario

En este apartado se pretende ilustrar, a modo de manual de usuario, la instanciación de una aplicación web basada en Puzzle.

El desarrollo básico de aplicaciones basadas en Puzzle comprende la creación de los componentes necesarios para realizar el ciclo de vida de las acciones dentro de la aplicación. Dichos componentes pueden ser generados a mano, pero para casi todos ellos existe un sistema de generación automática de código que, dependiendo del tipo de objeto en cuestión, genera un esqueleto o un objeto completo tal y como se ha descrito en los apartados anteriores dedicados al detalle de los generadores de componentes.

Es importante recordar que los generadores de Puzzle sobrescriben, por lo que aquellos elementos cuyo generador solo generé el esqueleto (todos salvo los DAOs, Eventos y EventosRespuesta) no deben ser regenerados una vez modificados, ya que se perderían todos los cambios.

Los detalles de cada uno de los pasos que se dan en el desarrollo de una aplicación Web basada en Puzzle, aquellos donde el framework se hace presente y se instancia, se ha descrito en el apartado 4.2.7-El proceso de desarrollo con Puzzle.

Una vez desarrollada la aplicación Web basada en Puzzle, se procederá a su distribución, instalación y despliegue, momento en el cual la aplicación está disponible para su uso.

6.1 Distribución, instalación y despliegue

La distribución, la instalación y el despliegue de una aplicación web basada en Puzzle no difiere del resto de las aplicaciones Web salvo la incorporación de las librerías y ficheros de configuración necesarios que hacen instanciable la aplicación web basada en Puzzle.

La arquitectura lógica de la aplicación se ilustra en la siguiente figura:

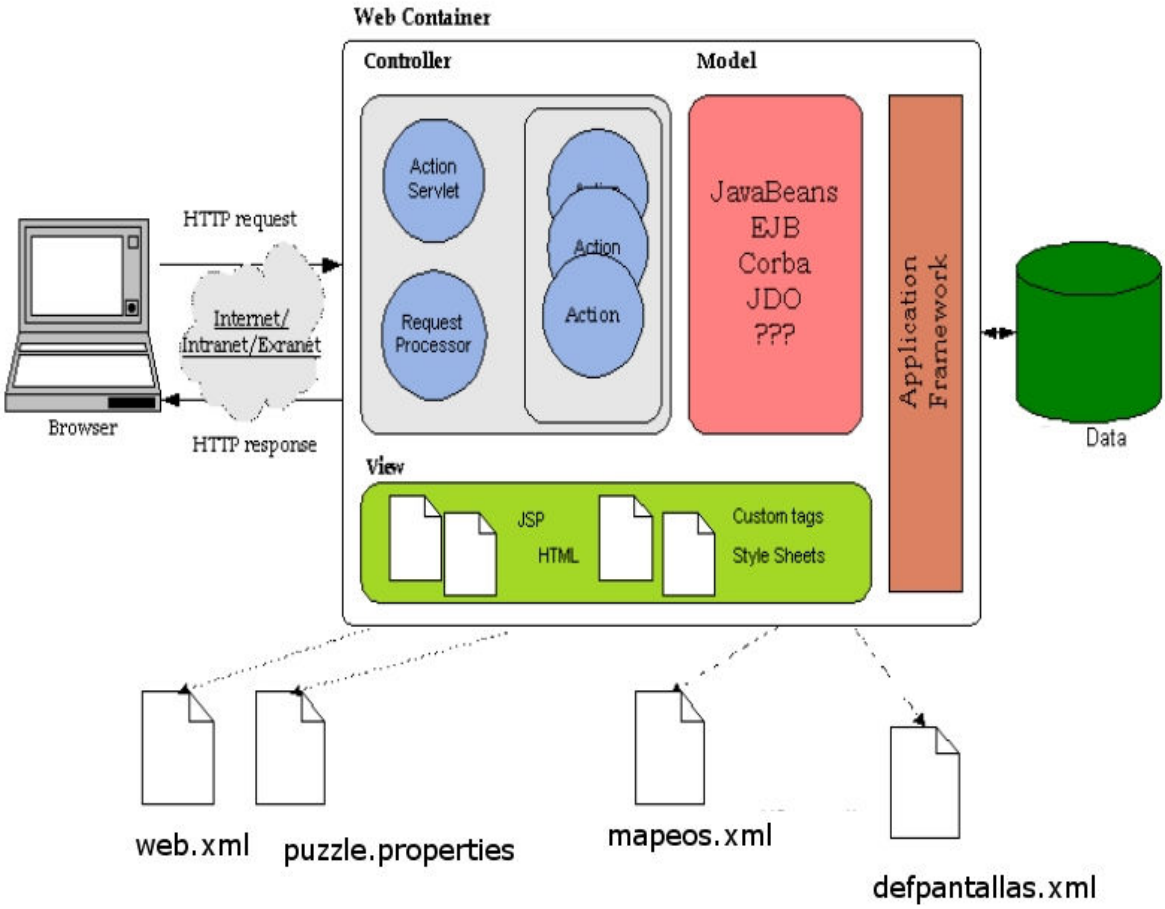

Ilustración 72 Arquitectura Lógica de una aplicación Web basada en Puzzle

Donde Application Framework corresponde a las librerías del núcleo del Framework Puzzle. Estas son:

- **Puzzle-web**: Contiene básicamente las implementaciones y componentes para el uso del framework (vista, controlador y modelo)
- **Puzzle-ejb**: Contiene básicamente las implementaciones y componentes para el uso del framework mediante EJBs (vista, controlador y modelo)
- **Puzzle-ejb-cliente**: Contiene básicamente las implementaciones y componentes para el uso del framework mediante EJB (controlador y modelo)

Una vez desarrollados los componentes y definidos los diferentes ficheros de configuración de la aplicación web, tal como se ha descrito en los apartados anteriores, se procedería al empaquetado de la aplicación web (fichero WAR o EAR, dependiendo del uso o no de componentes Enterprise como EJBs respectivamente) para su posterior distribución y despliegue. Esta actividad normalmente viene

incorporada en los entornos o IDE de desarrollo como por ejemplo el usado, ***IDE Eclipse***, o incluso se puede realizar manualmente con las utilidades incorporadas en el kit de desarrollo Java JDK.

La estructura de directorios resultante de ilustra en la siguiente figura:

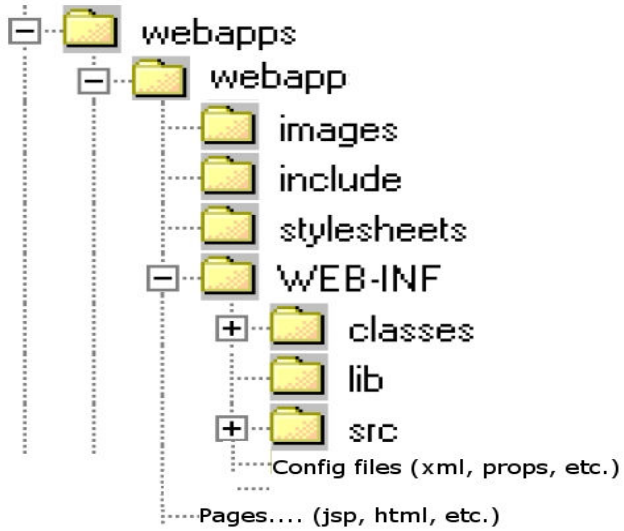

Ilustración 73 Estructura de directorios de una aplicación web

En última instancia, el despliegue, correspondería básicamente a la instalación o copia del fichero empaquetado (o en su defecto, todo el contenido) en el directorio de despliegue de aplicaciones correspondiente al Servidor Web o Aplicaciones destino.

Una vez hecho y rearrancado el servidor, automáticamente la aplicación web será desempaquetada y desplegada siendo accesible desde nuestro navegador web:

http://[host]:[port]/miWebApp/

7 Planificación

A continuación se va a explicar brevemente la planificación que se ha llevado a cabo para el proyecto, y que veremos resumido al final de la sección mediante un diagrama de Gannt.

El proyecto se ha dividido en cuatro fases:

- Análisis del sistema
- Diseño del sistema
- Implementación
- Memoria del Proyecto

En la primera fase, Análisis del Sistema, se estudió el sistema junto al entorno en el que se trabajaba. Una vez estudiado el sistema y su dominio, se comenzó a analizar los requisitos de usuario, los requisitos software y los casos de uso contrastándolo con los frameworks existentes en el mercado y también planificaciones internas de desarrollos similares. Éstos requisitos y casos de uso se correspondían con funcionalidades que ya existían así como las funcionalidades que se iban a implementar posteriormente.

En la fase de Diseño del Sistema, se crearon en primer lugar, mediante ingeniería inversa, todos los diagramas de clases de todos los componentes del sistema, para a continuación modificarlos, añadiendo las clases que eran necesarias para la implementación de las nuevas funcionalidades. Una vez terminados todos los diagramas de clases definitivos, se realizó la documentación de todos los componentes del Sistema.

La fase de implementación, se compone principalmente de dos subfases: Implementación de las partes del núcleo del framework propiamente dichas y por otro lado, las partes correspondientes a los puntos calientes o configurables así como los generadores y otras facilidades básicas. De todas ellas, la primera es la más importante y en la que más tiempo se ha invertido de todo este proyecto. Entre las dos subfases, también se ha indicado una tercera llamada *Modificación del Sistema,* en la que se ha querido englobar distintos cambios, mejoras o corrección de errores que se han realizado del sistema, pero que no se han especificado como nuevas subtareas.

Por último, la última fase ha sido la de la realización de la memoria, aunque en apartados como el Análisis del Sistema, o el Diseño del Sistema, ya se habían realizado y lo único que se ha tenido que hacer es adecuarlos al formato de la memoria. Como subfases se han puesto la redacción del *Estado del arte*, y *Resto de apartados* (introducción, conclusión, líneas futuras, etc.). Para terminar, las últimas dos subfases se refieran a la unión y el formateado de todos los apartados de la memoria.

En la siguiente ilustración se muestra el diagrama de Gannt donde se pueden observar gráficamente cada una de las fases y subfases que se acaban de explicar

Id	Nombre de tarea	Duración	Comienzo	Fin
1	**Inicio del Proyecto**	**299 días**	**lun 15/01/07**	**jue 06/03/08**
2	**Análisis del Sistema**	**46 días**	**lun 15/01/07**	**lun 19/03/07**
3	Estudio del sistema	21 días	lun 15/01/07	lun 12/02/07
4	Obtención de requisitos del Siste	11 días	mar 13/02/07	mar 27/02/07
5	Obtención de Casos de Uso del S	7 días	mié 28/02/07	jue 08/03/07
6	Arquitectura del Sistema	3 días	vie 09/03/07	mar 13/03/07
7	Modelo de datos del sistema	3 días	mié 14/03/07	vie 16/03/07
8	Documentación del Análisis	25 días	mar 13/02/07	lun 19/03/07
9	**Diseño del sistema**	**49 días**	**mar 20/03/07**	**vie 25/05/07**
10	Ingeniería inversa	6 días	mar 20/03/07	mar 27/03/07
11	Modificación Componentes	18 días	mié 28/03/07	vie 20/04/07
12	Definición de componentes	25 días	lun 23/04/07	vie 25/05/07
13	Documentación del diseño	38 días	mié 28/03/07	vie 18/05/07
14	**Implementación**	**176 días**	**lun 23/04/07**	**lun 24/12/07**
15	Funcionalidad rondas	98 días	lun 23/04/07	mié 05/09/07
16	Pruebas Rondas	3 días	jue 06/09/07	lun 10/09/07
17	Modificaciones Sistema	30 días	mar 11/09/07	lun 22/10/07
18	Integración de RMI al Sistema	20 días	jue 22/11/07	mié 19/12/07
19	Pruebas RMI	3 días	jue 20/12/07	lun 24/12/07
20	**Memoria del Proyecto**	**44 días**	**lun 07/01/08**	**jue 06/03/08**
21	Estado del arte	18 días	lun 07/01/08	mié 30/01/08
22	Resto de puntos	16 días	vie 01/02/08	vie 22/02/08
23	Unificacion puntos memoria	5 días	lun 25/02/08	vie 29/02/08
24	Formateado memoria	4 días	lun 03/03/08	jue 06/03/08

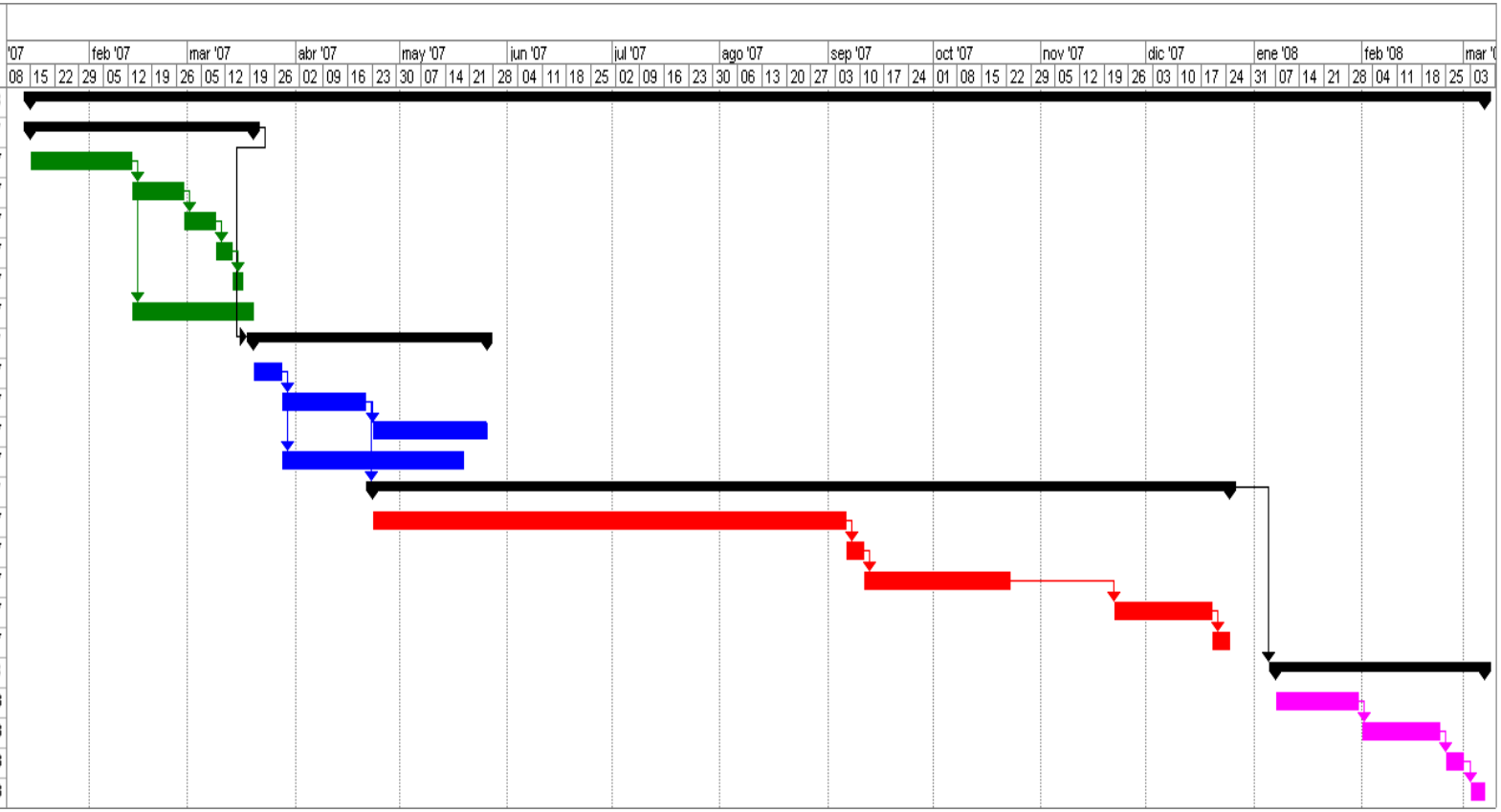

Ilustración 74 Diagrama de Gannt con la planificación del proyecto

8 Presupuesto

8.1 Recursos

Para la realización de este proyecto se han utilizado los recursos y costes descritos en los siguientes apartados.

8.1.1 Recursos software:

Eclipse IDE: **0 €**.

Licencia Servidores Web (Tomcat, JBoss, SunOne) para Telefónica I+D: **0 €**.

Licencia Oracle 9i para Telefónica I+D: **0 €.**

Licencia Microsoft Office 2007: **200 €**.

8.1.2 Recursos hardware:

Ordenador portátil (parte proporcional al proyecto): 4**00 €**.

8.1.3 Recursos humanos:

Los costes de recursos humanos han sido calculados observando el sueldo medio de un Ingeniero Superior en Informática sin experiencia trabajando 8 horas al día.

Coste de un Ingeniero Superior Informático **20 €/hora**.

En el apartado 7 Planificación, se mostraba el tiempo que se ha invertido en cada una de las fases. Poniendo como media 3 horas al día. En la siguiente tabla vamos a ver el cómputo de horas.

Fase	Subfase	Horas	Importe (€uros)
Análisis del Sistema	Estudio del sistema	57	684
	Obtención de requisitos del Sistema	20	240
	Obtención de Casos de Uso del Sistema	15	180
	Obtención de Casos de Uso del Dominio	12	144
	Arquitectura del Sistema	12	144
	Documentación del Análisis	60	720
	Total Análisis del Sistema	176	2.112
Diseño del Sistema	Ingeniería Inversa	12	144
	Modificación componentes	27	324
	Definición de componentes	42	504
	Documentación del diseño	72	864
	Total Diseño del Sistema	153	1.836
Implementación	Funcionalidad Framework (Núcleo)	270	3.240
	Pruebas	9	108
	Modificaciones Sistema	72	864
	Integración del Sistema	45	540
	Pruebas Integración-Instanciación	9	108
	Total Implementación	405	4.860
Memoria	Estado del Arte	48	576
	Resto de apartados	48	576
	Unificación apartados memoria	12	144
	Formateado memoria	9	108
	Total Memoria	117	1.404
	Total	851	10.212

Tabla 4 Recursos humanos del proyecto

Por tanto el coste total del proyecto ha sido:

Recursos	Importe (€uros)
Recursos software	200
Recursos hardware	580
Recursos humanos	17.212
Total	17.992

9 Futuras líneas de trabajo: Análisis de Viabilidad de Puzzle

9.1 Análisis del mercado interno y externo

9.1.1 Mercado interno

Una de las razones del nacimiento de Puzzle fue la existencia de un gran número de proyectos o aplicativos internos que se encontraban dentro del mismo dominio y solucionaban el mismo problema (tanto de infraestructura como de plataforma) cada uno de una manera. La homogeneización de todos estos proyectos se consigue con Puzzle y en particular, el uso del Puzzle en todos los proyectos web dentro de la división 2731, donde nació Puzzle. Estos son:

- Hércules Hospitalización (2731): Aplicación Web para la gestión Clínica de Pacientes
- Hércules Emergencias (2731): Aplicación Web con el mismo cometido que la anterior pero orientada a dispositivos móviles (TabletPC) y pensada para las Unidades Móviles de Emergencias o Primera Atención.
- SGIR: Sistema de Gestión de Reclamaciones y Incidencias (Telefónica Brasil)
- Portal del Distribuidor PK
- SIAM-U / Renovación de GRI
- PARESS
- Plataforma myJava.es (antiguamente denominada DG1000)

Dentro de cada proyecto, el uso de la plataforma es completo o parcial. La siguiente tabla ilustra, para cada proyecto, el uso de Puzzle en su sistema y donde subrayamos aquellos que forman parte del framework (el resto, facilidades incorporadas):

	HERCULES Hospitalización	HERCULES Emergencia	Portal Distribuidor (PK)	SGIR	PARESS	SIAM-U	Prototipo HTML SAM	mijava.es
Capa presentación								
Comp.presentación								

Capa controlador: web	■	■		■	■			
Capa negocio	■	■			■			
Capa de acceso a datos	■	■	■		■	■		■
Generador gráficos	■							
Generador documentos	■	■			■			
Gestor transaccional	■	■			■			
Svc. Admin. y monitor.	■	■						■
Svc. Firma	■							
Servicio seguridad	■	■						
Servicio JMS								
Servicio auditoría		■						
Generador Puzzle	■	■	■		■	■		■

Tabla 5 Mercado interno: Uso actual de Puzzle

En general, Puzzle se ha utilizado hasta ahora como un mecanismo de conseguir eficiencia en proyectos con presupuesto y tiempo escaso (como por ejemplo, el Portal Distribuidor y PARESS) más que como un mecanismo de obtener ingresos. SIAM-U se presenta como la primera ocasión en que se han solicitado licencias y la obtención de un beneficio por el uso de la plataforma.

Observamos que el producto es una plataforma viable y sobre todo beneficiosa, vislumbrando positivamente para el futuro la existencia de posibilidades de mercado interno. Para ello, el éxito se alcanzaría teniendo en cuenta las siguientes consideraciones:

- La necesidad de un adecuado soporte y publicidad, ya que se observa que el mercado interno posible de Puzzle es **amplísimo**.
- Detección de Proyectos J2EE para infraestructuras que requieren una reingeniería profunda:
  - Renovación tecnológica de GRI
  - Proyectos en el entorno de PARESS
  - Evolución de Portal del Distribuidor

- Proyectos para otras empresas del grupo con funcionalidad similar. Por ejemplo, TeleSP tomando como base SGIR.

Observamos que, en general, se puede utilizar en todos los proyectos que utilicen total o parcialmente J2EE. Incluso como se ha descrito anteriormente, algunos componentes se pueden utilizar fuera de entorno J2EE tales como las librerías JavaScript para la capa de presentación y el DAO para el acceso a bases de datos.

9.1.2 Mercado externo

Al igual que se demuestra la viabilidad de la plataforma dentro de los proyectos internos, se plantean las **posibilidades** de un mercado externo como producto. En este sentido, habría que analizar los siguientes factores según el sector o mercado:

- Se pueden considerar dos tipos de mercados:
  - Puzzle como plataforma
  - Aplicaciones construidas con Puzzle

- Los clientes naturales de la plataforma son:
  - Consultoras
  - Integradores
  - Comunidades de SW libre

- Los clientes de las aplicaciones son todo tipo de empresas e instituciones

Y dentro de los diferentes sectores que se pueden abordar, nos encontramos también con la elección de la estrategia. Las posibilidades que tenemos son las siguientes:

- **Opción 1 - SW LIBRE**: Se abre y publicita Puzzle y se forma una comunidad.
  - Debería alinearse con la estrategia general de software libre de TID

- **Opción 2 - PLATAFORMA COMERCIALIZABLE**: Generación de un producto comercializable como plataforma directamente.
  - No se contempla actualmente
  - Si se abordase, implicaría:
    - Afinar el producto (documentación, distribución, etc)
    - Definición de un claro esquema de precios
    - Acciones de marketing y distribución

- **Opción 3 - LICENCIAS**: Incorporación de coste de licencias en desarrollos de TID que la incorporen.
  - Los ingresos contribuirían a su mantenimiento

  - Necesario definir una clara política de licencias
  - Debería alinearse con la política general de la plataforma

- **Opción 4 - NO COMERCIALIZACIÓN**: Se utilizaría Puzzle como un elemento de mejora de la competitividad de TID pero sin ingresos directos.

La opción a corto plazo parece ser la de licencias (opción 3) pero deberían estudiarse otros esquemas y alinear estrategias.

Actualmente existe una aplicación Piloto, Historia Clínica de Emergencias (HERCULES Emergencias) para el cliente JCyL, aún en Certificación, y que al ser una aplicación piloto no va a revertir ingreso alguno. En el caso del aplicativo HERCULES, los ingresos vendrían eventualmente en el caso de despliegues (no de pilotos) y a través de licencias HERCULES, no Puzzle.

Podemos observar por tanto, que las posibilidades del mercado son muy buenas, en particular en administraciones públicas, donde en general J2EE está muy aceptado. Además, se tiene especial interés en el sector sanitario por la existencia del producto HERCULES y por la constancia de deseo de arquitectura J2EE, donde nos encontramos los siguientes ejemplos:

- Sistema Información Hospitalario Murcia (5 M€)
- JARA (26,5 M€)
- Otros

En principio, Puzzle se presenta con factible y aplicable en cualquier sector donde Telefónica esté o vaya a entrar dentro del dominio que cubre el framework.

9.2 Comparativa con productos existentes en el mercado

9.2.1 Planteamiento de la comparación

Antes de comenzar con la comparativa, vamos a plantear las premisas tomadas como marco de referencia para la comparación. Estas son:

- Respecto a la evaluación de herramientas del mercado:
  - Comparar con frameworks de planteamiento **amplio** tales como:
    - Struts
    - RealMethods
    - Spring Framework
    - Open Symphony

  - Comparar con Herramientas / componentes / estándares que ataquen funcionalidades **concretas** contempladas en Puzzle:
    - Dynapi
    - JDO
    - JFreeChart

  - XSL-FO
  - JFreeReport
  - Data Vision
  - JasperReports
  - Open Reports

- Énfasis en las herramientas de software libre
- La comparación se realiza componente a componente

En los siguientes apartados, se describen pormenorizadamente cada una de los elementos listados para realizar la comparación y mediante matrices de trazabilidad, donde se compara la aportación que realiza Puzzle contrastándolo con las citadas herramientas.

9.2.2 Tablas comparativas

Una breve descripción de los frameworks que vamos a comparar con Puzzle se ilustra en la siguiente tabla:

Framework	Descripción
STRUTS	■ Framework Open source para implementación aplicaciones web ■ Modelo MVC Model 2 (solo vista y controlador, no MODELO) ■ Extensible (Tiles, Struts Workflow,...) ■ De amplia utilización y conocimiento ■ Buena documentación
realMethods	■ Framework comercial ■ Usa patrones de diseño ■ Capaz de integrarse con Struts ■ Arquitectura MDA ■ Acceso a datos vía DAO

Spring Framework	■ **Framework Open source de propósito general para aplicaciones J2EE o J2SE** ■ **Paradigma IoC (Inversion of Control)->El componente desarrolla JavaBeans que son gestionados por el framework** ■ **Supone un paradigma de programación completamente diferente (programación orientada a aspectos)** ■ **Genera muchos objetos en tiempo de ejecución**
Open Symphony – WebWork	■ **Proyecto Open Source** ■ **Se centra en capas vista y controlador** ■ **Integración con distintas tecnologías de presentación** ■ **Muy similar a Struts**

Tabla 6 Algunos frameworks libres del mercado

Por otro lado, los componentes o estándares concretos a comparar son:

Solución	Descripción
Dynapi	Desarrollo Open Source para crear componente visuales avanzados. No se encuentra integrada en ningún framework.
JDO	Estándar de acceso a datos que permite abstraerse de la tecnología bajo la que se encuentran dichos datos. Se integra bien con J2EE y J2SE.
JFreeChart	Herramienta open source para generación de gráficos. Soporta exportación a PDF y SVG.
XSL-FO	Estándar del W3C que permite definir la conversión de un documento XML en un formato de salida. Existe alguna herramienta que lo implementa (FOP, Apache)
JFreeReport	Herramienta Open Souce que permite a partir de XML generar PDF, HTML, CSV, Excel y texto plano)
DataVision	Herramienta Open source de generación de informes similar a Crystal Report que genera informes en PDF, XML, HTML, LaTeX2e, DocBook, etc
Jasper Reports	Herramienta de software libre del mismo estilo que JFreeReport
Open reports	Desarrollo realizado sobre Jasper Reports que incluye un editor para diseño de informes.

Tabla 7 Algunos componentes / estándares concretos del mercado

El resultado de la comparación, describiendo las funcionalidades que cubre Puzzle con respecto al resto de productos del mercado libre, se ilustra en la siguiente tabla:

	PUZZLE	Struts	realMethods	Spring Framework	Open Symphony	Dynapi	JDO (java Data Object)	JFreeChart	XSL-FO	JFreeReport	Data Vision	JasperReports	Open reports
Capa presentación													
Comp.presentación													
Capa controlador: web													
Capa negocio													
Capa de acceso a datos													
Generador gráficos													
Generador documentos													
Gestor transaccional													
Svc. Admin. y monitor.													
Svc. Firma													
Servicio seguridad													
Servicio JMS													

Servicio auditoría													
Generador													

Tabla 8 Tabla comparativa por componentes

9.2.3 Valoración global

A partir del estudio comparativo realizado obtenemos el siguiente resultado:

- **Puzzle vs. Frameworks**: Puzzle, en general, es una plataforma que formaliza todos los aspectos del desarrollo J2EE (otros frameworks como Struts u Open Symphony se centran en aspectos concretos). Además, Puzzle también incluye los componentes reutilizables de más alto nivel (facilidades) que, en general, no encontramos en otros frameworks. En cualquier caso, no hay que olvidar que algunos frameworks, ofrecen diseños o componentes muy interesantes y avanzados que habrá que tener presente en la evolución de la plataforma.

  Un caso particular es Struts, el cual goza de amplia aceptación y conocimientos y se utiliza, tanto individualmente como conjuntamente, en proyectos de TID, de ahí, que se ofrezca una fácil integración entre estos dos framework, tal y como ilustra la siguiente figura:

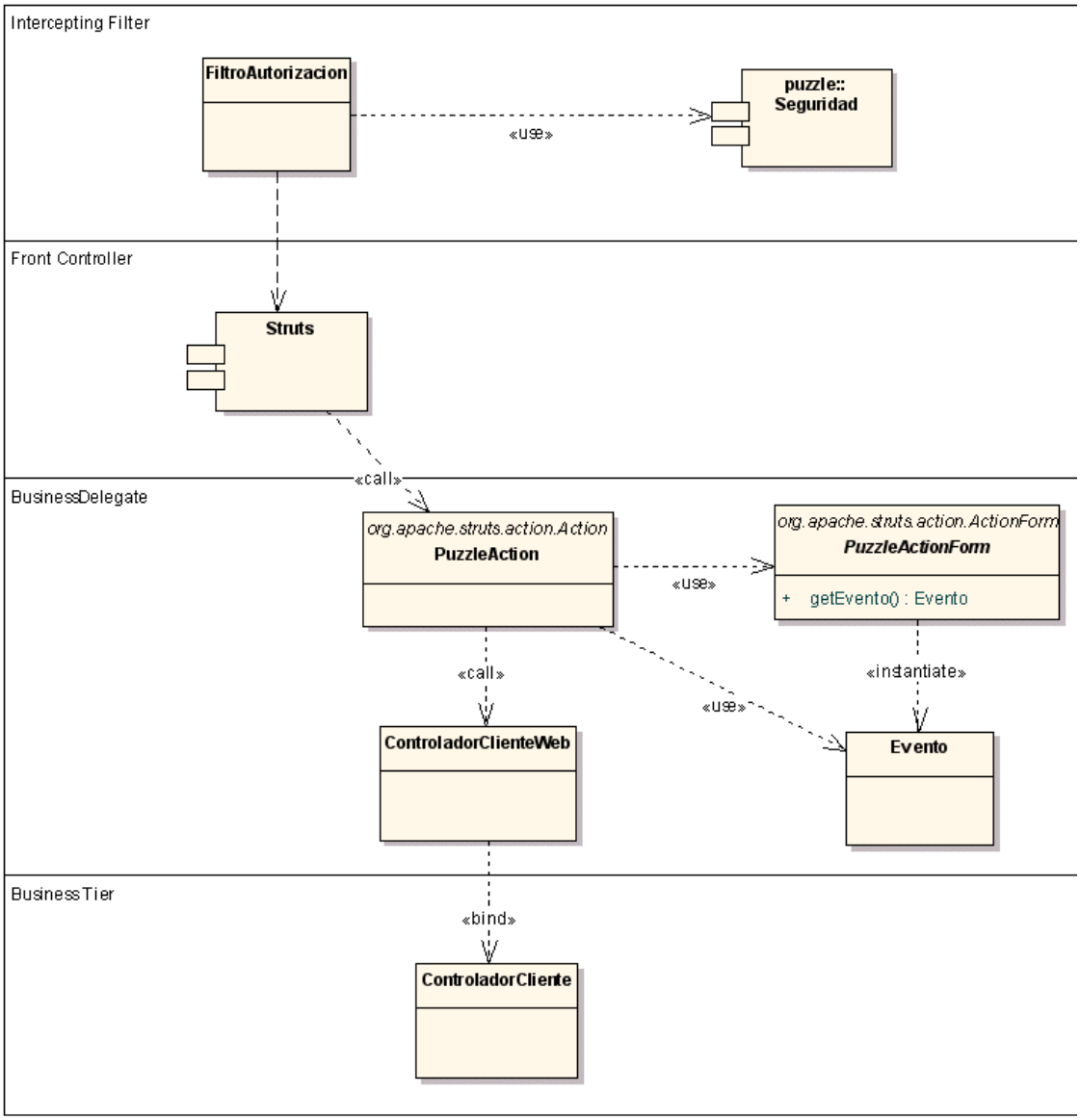

Ilustración 75 Esquema de integración con Struts

Observamos que se delega la capa de presentación y controlador a Struts y se usa el framework Puzzle para cubrir precisamente la carencia que éste primero ofrece, la capa de negocio.

- **Puzzle vs. Elementos específicos**: Existen algunos componentes interesantes y, en general, los componentes estudiados se integrarían bien con Puzzle.

9.2.3.1 Estrategias frente a herramientas del mercado

Una vez obtenida una valoración global en la comparativa de Puzzle como plataforma con las herramientas existentes en el mercado, se proponen la distintas estrategias para abordar proyectos web según el framework y/o el componente. Estas son:

- **Estrategia 1 - INCORPORACIÓN**: Cuando la herramienta del mercado es compatible con Puzzle y la enriquece. Por ejemplo:
  - Taglibs STRUTS o JSTL
  - Herramienta de generación de gráficos (JFreeChart)

- **Estrategia 2 - CONVIVENCIA**: Cuando la herramienta del mercado es parcialmente complementaria/competidora de Puzzle, tiene suficiente fuerza en el mercado pero es posible la convivencia. Ejemplo:
  - STRUTS
  - Herramientas generación documentos (JFreeReport)

- **Estrategia 3 - COMPETENCIA**: Cuando no existe algo análogo en el mercado, la herramienta de mercado es comercial o Puzzle es claramente superior.

9.2.4 Líneas básicas de evolución de Puzzle

Teniendo en cuenta el mercado y necesidades en continuo cambio en cuanto a los servicios y aplicaciones web se refiere, y teniendo en mente el objetivo de que la plataforma Puzzle sea útil en todo momento adaptándose a las nuevas necesidades, se tienen que definir unas líneas básicas de evolución de la plataforma para cumplir el objetivo. En líneas generales, estas son las siguientes:

- **INTEGRACIÓN EN PLATAFORMA Y FORMACIÓN DE LA "COMUNIDAD PUZZLE":** Generación de la infraestructura necesaria para que Puzzle pueda ser utilizada en TID de manera global, proporcionando servicios de plataforma y comunidad. La comunidad se presenta como un factor fundamental para asegurar la longevidad de la plataforma.
  - Documentación
  - Web
  - Foro
  - Soporte
- **EVOLUCIÓN DEL NÚCLEO PUZZLE**: Evolución de Puzzle para incorporar mejoras en todos sus elementos con independencia de otros productos. Podemos agruparlo en los siguientes contextos:
  - **Evolución del Framework**: Aseguramiento de la compatibilidad con normativas y referencias, en particular para los proyectos de TID, compatibilidad con normativa OSI. Otros factores también de interés como la evolución del DAO y otras mejoras de más detalle.
  - **Evolución de las Facilidades:** Como el aislamiento del framework, aumentar la cohesión y disminuir el acoplamiento y, en general, abstracción de las facilidades.
  - **Evolución de las Herramientas:** Mejorando los procesos de automatización, haciéndolos más intuitivos y manejables, por ejemplo, generadores con una interfaz gráfica.

- **INTEGRACIÓN CON OTRAS PLATAFORMAS**: Como se ha descrito antes, también es fundamental que existan desarrollos orientados a posibilitar la convivencia con otros frameworks o plataformas que se consideran de interés. (aplicación de la estrategia de convivencia, por ejemplo, con Struts)

- **INCORPORACIÓN COMPONENTES:** Evaluación, prueba e incorporación de componentes procedentes del software libre (aplicación de estrategia de incorporación). Entre los ejemplos citados, nos podemos encontrar:
    - Tags avanzadas para la generación de vistas (TagLibs de Struts)
    - Generación de gráficos (basándose en JFreeChart.
    - Generación de documentos basándose en JFreeReport

10 Conclusiones

10.1 Objetivo

Un enfoque a los frameworks debe considerar cuando los requisitos del producto cambian rápidamente. Los frameworks se pueden también utilizar para aumentar el desarrollo; implementando al principio un código hot-spot simple y posteriormente actualizarlo. Una buena referencia de frameworks es [4], aquella que hace una buena evaluación de su metodología y sus últimos avances.

El tema de los frameworks es algo que aún sigue en desarrollo e investigación. En consecuencia, aún quedan muchos problemas por resolver, tales como la documentación del framework. La falta de un terreno común crea un espacio vacío entre los desarrolladores de frameworks, los extendedores y los usuarios. La parte económica es otro de los problemas en la implementación de los frameworks el cual hay que estimar el costo de construir un framework versus las aplicaciones, y por consiguiente, las utilidades que generaría esa inversión.

Finalmente creemos que los recién llegados al escenario de desarrollo del framework, debieran analizar los puntos aquí expuestos con sumo cuidado, considerando lo que el actor necesita, que es lo que está haciendo, y ser consciente de que los frameworks tienen sus pros y sus contras, debido a que no son la solución para todos los problemas. Además, siempre hay que tener presente, que si alguien está considerando usar un framework ya existente, debe de estudiar su documentación y verificar si hay ejemplos de instanciación que demuestran su viabilidad. Es importante también observar, dentro de lo posible, las aplicaciones que fueron generadas y la cantidad de esfuerzo dedicado a este proceso.

Por otro lado, el objetivo del proyecto se ha logrado, demostrando el éxito del mismo tal y como se describe en el apartado de viabilidad de la plataforma, demostrando su objetivo en los innumerables proyectos que ya usan la plataforma y los futuros proyectos que la usarán.

Finalmente no hay que olvidar a las personas con las que he trabajado estrechamente, codo con codo, para conseguir que este proyecto de plataforma e innovación de Telefónica I+D se convirtiera en realidad, tales como Ignacio González de los Reyes Gavilán, como Jefe de División que apostó y creyó en el proyecto cuando se lo propusimos, apoyándonos en todo momento durante las primeras fases de la plataforma, a José Ángel Gómez Diez, el que me acompañó desde el primer momento en este proyecto, de hecho, nos podemos considerar como los creadores de lo que ahora es la Plataforma Puzzle, y por supuesto, sin olvidar a otros como José Luis Martínez Ávila o Inmaculada García Alarcón que se encargaron de la documentación y la incorporación de las nuevas necesidades y facilidades; también a Rubén Alonso Alejandre como creador de todas las librerías de apoyo a la presentación, o a Manuel José Hernández Gajate como principal tomador del testigo de las sucesivas evoluciones de Puzzle y las mejoras que las distintas aplicaciones iban requiriendo, en definitiva, al Equipo de Puzzle que ha ido creciendo a lo largo de los años dentro de la división 2731.

10.2 Futuro

Como se ha descrito en la viabilidad del proyecto, está en un proceso continuo de mejora y de evolución. Aunque no estoy formando parte del desarrollo de las nuevas mejoras, facilidades y en general, nuevas funcionalidades de la plataforma, sigo colaborando estrechamente en su evolución correctiva y adaptativa, tanto en el propio núcleo del framework como en el apoyo, consultoría y asesoría de los proyectos instanciados basados en Puzzle.

Anexo A Web Deployment Descriptor versión 2.3

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC"-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
        "http://java.sun.com/dtd/web-app_2_3.dtd">

<web-app>

<display-name>PARESS</display-name>

        <listener>

                <listener-class>tid.paress.controlador.web.LocalizadorServiciosParessTomcat

</listener-class>

</listener>

<servlet>

        <servlet-name>ServletControlador</servlet-name>

        <display-name>ServletControlador</display-name>

        <description>Controlador Frontal del Sistema entorno Web</description>

<servlet-class>tid.puzzle.framework.controlador.web.ServletControlador</servlet-class>

</servlet>

<servlet>

        <servlet-name>ServletPlantilla</servlet-name>

        <display-name>ServletPlantilla</display-name>

        <servlet-class>tid.puzzle.framework.vista.plantilla.ServletPlantilla</servlet-class>

        <init-param>

                <param-name>lenguaje_por_defecto</param-name>

                <param-value>sp</param-value>

        </init-param>

</servlet>

<servlet-mapping>

        <servlet-name>ServletControlador</servlet-name>

        <url-pattern>*.accion</url-pattern>

</servlet-mapping>

<servlet-mapping>

        <servlet-name>ServletPlantilla</servlet-name>

        <url-pattern>*.pantalla</url-pattern>

</servlet-mapping>
```

```
<session-config>
        <session-timeout>30</session-timeout>
</session-config>
<welcome-file-list>
        <welcome-file>index.htm</welcome-file>
</welcome-file-list>
<resource-env-ref>
        <description>DB Connection</description>
        <resource-env-ref-name>TIDPool</resource-env-ref-name>
        <resource-env-ref-type>javax.sql.DataSource</resource-env-ref-type>
</resource-env-ref>
<security-constraint>
        <web-resource-collection>
                <web-resource-name>Jsp's</web-resource-name>
                <description>Ninguna JSP se puede invocar de forma directa</description>
                <url-pattern>*.jsp</url-pattern>
                <url-pattern>*.pantalla</url-pattern>
                <http-method>POST, GET</http-method>
        </web-resource-collection>
        <auth-constraint>
                <role-name>CONTAINER</role-name>
        </auth-constraint>
</security-constraint>
</web-app>
```

Anexo B Web Deployment Descriptor versión 2.2

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!DOCTYPE web-app PUBLIC"-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
        "http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">

<web-app>

<display-name>PARESS</display-name>

<servlet>

        <servlet-name>ServletControlador</servlet-name>

        <display-name>ServletControlador</display-name>

        <description>Controlador Frontal del Sistema entorno Web</description>

<servlet-class>

tid.puzzle.framework.controlador.web.ServletControlador

</servlet-class>

</servlet>

<servlet>

        <servlet-name>ServletPlantilla</servlet-name>

        <display-name>ServletPlantilla</display-name>

        <servlet-class>tid.puzzle.framework.vista.plantilla.ServletPlantilla

</servlet-class>

        <init-param>

                <param-name>lenguaje_por_defecto</param-name>

                <param-value>sp</param-value>

        </init-param>

</servlet>

<servlet>

        <servlet-name>ServletSessionListener</servlet-name>

        <display-name>ServletSessionListener</display-name>

        <description>Emulacion SessionListener</description>

        <servlet-class>tid.puzzle.framework.controlador.web.ServletSessionListener

</servlet-class>

        <init-param>

                <param-name>servlet</param-name>

                <param-value>ServletControlador</param-value>
```

```
            </init-param>
            <init-param>
                  <param-name>controladorUsuarioWeb</param-name>
                  <param-value>tid.paress.controlador.web.ControladorUsuarioWeb</param-value>
            </init-param>
</servlet>
<servlet-mapping>
            <servlet-name> ServletSessionListener</servlet-name>
            <url-pattern>*.accion</url-pattern>
</servlet-mapping>
<servlet-mapping>
            <servlet-name>ServletPlantilla</servlet-name>
            <url-pattern>*.pantalla</url-pattern>
</servlet-mapping>
<!-- El resto del descriptor es igual que el de la versión 2.3 -->
</web-app>
```

Anexo C Ejemplo Mapeos.xml (Hercules WebApp)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<mapeos>

	<pantalla_por_defecto>principal.pantalla</pantalla_por_defecto>

	<!-- Ventana de login -->

	<mapeo_url url="login.accion" pantalla="login.pantalla" esAccion="true" usaManejadorFlujo="false">

		<clase_accion>tid.hercules.controlador.web.accion.WebAccionSalidaApp</clase_accion>

	</mapeo_url>


	<!-- ENTRADA Y SALIDA DEL SISTEMA -->

	<!-- Entrada en el Sistema -->

	<mapeo_url url="entrada.accion" pantalla="principal.pantalla" esAccion="true"
usaManejadorFlujo="true">

		<clase_accion>tid.hercules.controlador.web.accion.WebAccionLoginCCU</clase_accion>

		<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorFlujoEntradaPrincipal">

			<resultado_manejador resultado="0" pantalla="loginerror.pantalla"/>

			<resultado_manejador resultado="ccu" pantalla="principalCCU.pantalla"/>

			<resultado_manejador resultado="gerencia" pantalla="principalGerencia.pantalla"/>

			<resultado_manejador resultado="admon" pantalla="principalAdmon.pantalla"/>

		</manejador_flujo>

	</mapeo_url>


	<!-- Salida del sistema -->

	<mapeo_url url="exitSesion.accion" pantalla="login.pantalla" esAccion="true"
usaManejadorFlujo="false">

		<clase_accion>tid.hercules.controlador.web.accion.WebAccionSalidaApp</clase_accion>

	</mapeo_url>

	<mapeo_url url="exit.accion" pantalla="exit.pantalla" esAccion="true" usaManejadorFlujo="false">

		<clase_accion>tid.hercules.controlador.web.accion.WebAccionSalidaApp</clase_accion>

	</mapeo_url>


	<!-- FIN ENTRADA Y SALIDA DEL SISTEMA -->


	<!-- Acceso a la pantalla principal del CCU -->
```

```
<mapeo_url url="principal.accion" pantalla="principal.pantalla" esAccion="false" usaManejadorFlujo="true">

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorFlujoEntradaPrincipal">

<resultado_manejador resultado="ccu" pantalla="principalCCU.pantalla"/>

<resultado_manejador resultado="gerencia" pantalla="principalGerencia.pantalla"/>

<resultado_manejador resultado="admon" pantalla="principalAdmon.pantalla"/>

</manejador_flujo>

</mapeo_url>


<!-- Cambio de password -->

<mapeo_url url="cambiarPassword.accion" pantalla="cambiarPassword.pantalla" esAccion="false" usaManejadorFlujo="false"/>


<!-- Wizard Diagnostico -->

<mapeo_url url="navegaWizardDiagnostico.accion" pantalla="wizardDiagnostico.pantalla" esAccion="false" usaManejadorFlujo="true">

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManFlujoNavegaWizardDiagnostico">

<resultado_manejador resultado="1" pantalla="wizardDiagnostico.pantalla"/>

<resultado_manejador resultado="2" pantalla="error.pantalla"/>

</manejador_flujo>

</mapeo_url>


<!-- GENERADOR DE DOCUMENTOS -->

<mapeo_url url="generarDocumento.accion" pantalla="principal.documento" esAccion="true" usaManejadorFlujo="true">


<clase_accion>tid.hercules.controlador.web.accion.WebAccionGenerarDocumento</clase_accion>

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorFlujoGenerarDocumento">

<resultado_manejador resultado="1" pantalla="principal.documento"/>

</manejador_flujo>

</mapeo_url>

<!-- FIN GENERADOR DE DOCUMENTOS -->


<!-- PACIENTES -->
```

```
<mapeo_url url="navegaNuevoPaciente.accion" pantalla="nuevoPaciente.pantalla" esAccion="false" usaManejadorFlujo="false"/>

<!-- Alta/ Modificación paciente -->

<mapeo_url url="insertarPaciente.accion" pantalla="principal.pantalla" esAccion="true" usaManejadorFlujo="true">


<clase_accion>tid.hercules.controlador.web.accion.WebAccionInsertarPaciente</clase_accion>

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorFlujoInsertarPaciente">

<resultado_manejador resultado="1" pantalla="principal.pantalla"/>

<resultado_manejador resultado="2" pantalla="insertarPacienteError.pantalla"/>

</manejador_flujo>

</mapeo_url>

<mapeo_url url="navegaLocPaciente.accion" pantalla="locPaciente.pantalla" esAccion="false" usaManejadorFlujo="false"/>

<mapeo_url url="navegaCstDatosPersonalesPaciente.accion" pantalla="listaDatosPersonalesPaciente.pantalla" esAccion="false" usaManejadorFlujo="true">

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorDatosPersonalesPaciente">

<resultado_manejador resultado="1" pantalla="listaDatosPersonalesPaciente.pantalla"/>

</manejador_flujo>

</mapeo_url>

<mapeo_url url="entradaMantPaciente.accion" pantalla="consultaPaciente.pantalla" esAccion="false" usaManejadorFlujo="true">

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorFlujoConsultaPaciente">

<resultado_manejador resultado="1" pantalla="consultaPaciente.pantalla"/>

</manejador_flujo>

</mapeo_url>

<mapeo_url url="editaPaciente.accion" pantalla="modPaciente.pantalla" esAccion="false" usaManejadorFlujo="true">

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorFlujoMantPaciente">

<resultado_manejador resultado="1" pantalla="modPaciente.pantalla"/>

</manejador_flujo>

</mapeo_url>

<mapeo_url url="mantPaciente.accion" pantalla="modPaciente.pantalla" esAccion="true" usaManejadorFlujo="true">


<clase_accion>tid.hercules.controlador.web.accion.WebAccionInsertarPaciente</clase_accion>

<manejador_flujo clase="tid.hercules.controlador.web.flujo.ManejadorFlujoMantPaciente">
```

```
                    <resultado_manejador resultado="1" pantalla="modPaciente.pantalla"/>

                </manejador_flujo>

            </mapeo_url>

            <mapeo_url url="terminarPacientes.accion" pantalla="principal.pantalla" esAccion="false"
usaManejadorFlujo="true">

                <manejador_flujo
clase="tid.hercules.controlador.web.flujo.ManejadorFlujoTerminarPacientes">

                    <resultado_manejador resultado="1" pantalla="principal.pantalla"/>

                </manejador_flujo>

            </mapeo_url>

            <!-- FIN PACIENTES -->

 [....]

</mapeos>
```

Anexo D Ejemplo DefPantallas_sp.xml (Hercules WebApp)

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<!--
 Copyright
 Grupo de Desarrollo de Puzzle TID
-->

<definicion_pantallas>
 <plantilla>/plantilla.jsp</plantilla>

 <!-- ********* -->
 <!--  GENERAL  -->
 <!-- ********* -->
 <pantalla>
  <nombre>principalCCU</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/>
  <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Incidentes abiertos" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/emergencias/incidentes/listaIncidentesAbiertos.jsp"
inclusion_directa="false"/>
  <parametro clave="botonera" valor="/emergencias/incidentes/botoneraCCU.jsp" inclusion_directa="false" />
 </pantalla>

 <pantalla>
  <nombre>principalGerencia</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/>
  <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
```

```
  <parametro clave="tituloPantalla" valor="Incidentes abiertos" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/emergencias/incidentes/listaIncidentesAbiertos.jsp" inclusion_directa="false"/>
  <parametro clave="botonera" valor="/emergencias/incidentes/botoneraGerencia.jsp" inclusion_directa="false" />
 </pantalla>

 <pantalla>
  <nombre>principalAdmon</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/>
  <parametro clave="cuerpo" valor="cuerpo.jsp" inclusion_directa="false"/>
 </pantalla>

 <pantalla>
  <nombre>error</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <!--parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/-->
  <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Error" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/error.jsp" inclusion_directa="false"/>
 </pantalla>

 <pantalla>
  <nombre>exito</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <!--parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/-->
  <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Exito" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/error.jsp" inclusion_directa="false"/>
 </pantalla>
```

```
<pantalla>
 <nombre>excapp</nombre>
 <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
 <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
 <!--parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/-->
 <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
 <parametro clave="tituloPantalla" valor="Error" inclusion_directa="true"/>
 <parametro clave="contenidoPantalla" valor="/excepcion.jsp" inclusion_directa="false"/>
</pantalla>

<pantalla>
 <nombre>excseg</nombre>
 <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
 <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
 <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
 <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
 <parametro clave="tituloPantalla" valor="¡Atención!" inclusion_directa="true"/>
 <parametro clave="contenidoPantalla" valor="/excepcionSeguridad.jsp" inclusion_directa="false"/>
</pantalla>

 <pantalla>
    <nombre>login</nombre>
    <parametro clave="titulo" valor="Hércules (Historia Clínica Electronica)" inclusion_directa="true" />
    <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false" />
    <parametro clave="marco_izda" valor=" " inclusion_directa="true" />
    <parametro clave="cuerpo" valor="/multilogin.jsp" inclusion_directa="false" />
 </pantalla>

 <pantalla>
    <nombre>loginTarjeta</nombre>
    <parametro clave="titulo" valor="Hércules (Historia Clínica Electronica)" inclusion_directa="true" />
    <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false" />
```

```
    <parametro clave="marco_izda" valor=" " inclusion_directa="true" />
    <parametro clave="cuerpo" valor="/loginTarjeta.jsp" inclusion_directa="false" />
  </pantalla>

<pantalla>
  <nombre>exit</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
  <parametro clave="cuerpo" valor="/exit.jsp" inclusion_directa="false"/>
</pantalla>

<pantalla>
  <nombre>loginexito</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/>
  <parametro clave="cuerpo" valor="/exito.jsp" inclusion_directa="false"/>
</pantalla>

<pantalla>
  <nombre>loginerror</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
  <parametro clave="cuerpo" valor="/pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Error de identificación" inclusion_directa="true" />
  <parametro clave="contenidoPantalla" valor="/loginerror.jsp" inclusion_directa="false" />
</pantalla>

<pantalla>
  <nombre>logincriticalerror</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
```

```
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
  <parametro clave="cuerpo" valor="/pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Error de identificación" inclusion_directa="true" />
  <parametro clave="contenidoPantalla" valor="/logincriticalerror.jsp" inclusion_directa="false" />
</pantalla>

<!-- PREFERENCIAS -->

<pantalla>
  <nombre>cambiarPassword</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/>
  <parametro clave="cuerpo" valor="/pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Cambiar password" inclusion_directa="true" />
  <parametro clave="contenidoPantalla" valor="/cambPass.jsp" inclusion_directa="false" />
</pantalla>

<pantalla>
  <nombre>preferencias</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
  <parametro clave="cuerpo" valor="/pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Preferencias de usuario" inclusion_directa="true" />
  <parametro clave="contenidoPantalla" valor="/preferencias.jsp" inclusion_directa="false" />
</pantalla>

<!-- *** FIN GENERAL *** -->

<!-- ********************** -->
<!-- MODULO PACIENTE (COMUN) -->
```

```
<!-- ********************** -->

<pantalla>
  <nombre>reasignarEpisodio</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/>
  <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Reasignación de Episodios" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/historiaclinica/reasignaEpisodio.jsp"
inclusion_directa="false"/>
</pantalla>

<pantalla>
  <nombre>locPaciente</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor="/menu.jsp" inclusion_directa="false"/>
  <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Búsqueda de Historia/Paciente" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/paciente/locPaciente.jsp" inclusion_directa="false"/>
</pantalla>

<pantalla>
  <nombre>listaDatosPersonalesPaciente</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
  <parametro clave="cuerpo" valor="/pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Lista de Pacientes" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/paciente/listaDatosPersonalesPaciente.jsp"
inclusion_directa="false"/>
  <parametro clave="destino" valor="entradaMantPaciente.accion" inclusion_directa="true"/>
  <parametro clave="botonera" valor="/paciente/botoneraListaPacientes.jsp" inclusion_directa="false" />
```

```
</pantalla>

<pantalla>
 <nombre>nuevoPaciente</nombre>
 <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
 <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
 <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
 <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
 <parametro clave="tituloPantalla" valor="Alta de Pacientes" inclusion_directa="true"/>
 <parametro clave="contenidoPantalla" valor="/paciente/nuevoPaciente.jsp" inclusion_directa="false"/>
 <parametro clave="botonera" valor="/paciente/botoneraAlta.jsp" inclusion_directa="false"/>
</pantalla>

<pantalla>
 <nombre>modPaciente</nombre>
 <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
 <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
 <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
 <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
 <parametro clave="tituloPantalla" valor="Mantenimiento de Pacientes" inclusion_directa="true"/>
 <parametro clave="contenidoPantalla" valor="/paciente/nuevoPaciente.jsp" inclusion_directa="false"/>
 <parametro clave="botonera" valor="/paciente/botoneraMod.jsp" inclusion_directa="false"/>
</pantalla>

<pantalla>
 <nombre>consultaPaciente</nombre>
 <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
 <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
 <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
 <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
 <parametro clave="tituloPantalla" valor="Consulta de Pacientes" inclusion_directa="true"/>
 <parametro clave="contenidoPantalla" valor="/paciente/consultaPaciente.jsp" inclusion_directa="false"/>
 <parametro clave="botonera" valor="/paciente/botoneraC.jsp" inclusion_directa="false"/>
```

```
</pantalla>

<pantalla>
    <nombre>datosPacienteHC</nombre>
    <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
    <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
    <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
    <parametro clave="cuerpo" valor="/pantallaHercules.jsp" inclusion_directa="false"/>
    <parametro clave="tituloPantalla" valor="Paciente" inclusion_directa="true"/>
    <parametro clave="contenidoPantalla" valor="/paciente/nuevoPaciente.jsp" inclusion_directa="false"/>
    <parametro clave="botonera" valor="/historiaclinica/botoneraModDatosHC.jsp" inclusion_directa="false"/>
</pantalla>

<!-- Pantalla de modificación de datos administrativos de pacientes desde anamnesis -->
<pantalla>
  <nombre>datosPacienteAnam</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
  <parametro clave="cuerpo" valor="/pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Paciente" inclusion_directa="true"/>
  <parametro clave="contenidoPantalla" valor="/paciente/nuevoPaciente.jsp" inclusion_directa="false"/>
  <parametro clave="botonera" valor="/historiaClinica/anamnesis/botoneraModDatosAnam.jsp"
inclusion_directa="false"/>
</pantalla>

<pantalla>
  <nombre>insertarPacienteError</nombre>
  <parametro clave="titulo" valor="HérCuLes (Historia Clínica Electrónica)" inclusion_directa="true"/>
  <parametro clave="cabecera" valor="/titulo.jsp" inclusion_directa="false"/>
  <parametro clave="marco_izda" valor=" " inclusion_directa="true"/>
  <parametro clave="cuerpo" valor="pantallaHercules.jsp" inclusion_directa="false"/>
  <parametro clave="tituloPantalla" valor="Error" inclusion_directa="true"/>
```

```
  <parametro clave="contenidoPantalla" valor="/paciente/insertarPacienteError.jsp"
inclusion_directa="false"/>

 </pantalla>



 <!-- *** FIN PACIENTES *** -->


 [...]


</definicion_pantallas>
```

11 Bibliografía

[1]. Booch, G., Jacobson, I., Rumbaugh, J., and Rumbaugh, J. The Unified Modeling Language User Guide. Addison-Wesley Pub Co, 1998.

[2]. Brooks, Jr., F. P. No Silver Bullet--Essence and Accidents of Software Engineering. IEEE Computer 20(4), April, 10-19, 1987.

[3]. CORBA Website. http://www.corba.org/.

[4]. Fayad, M. E., Schmidt, D. C., and Johnson, R. E. Building Application Frameworks. Addison-Wesley Pub Co, 1st edition, 1999.

[5]. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. Design Patterns : Elements of Reusable Object-Oriented Software. Addison-Wesley Pub Co, 1st edition, January 1995.

[6]. Jacobson, I., Booch, G., Rumbaugh, J. The Unified Software Developement Process. Addison-Wesley, 1999.

[7]. JUnit, Testing Resources for Extreme Programming, http://www.junit.org.

[8]. Mattsson, M. Object-Oriented Frameworks: A Survey of Methodological Issues. Technical Report 96-167, Dept. of Software Eng. and Computer Science, University of Karlskrona/Ronneby.

[9]. Mattsson, M., Bosch, J., and Fayad, M.E. Framework Integration Problems, Causes, Solutions. Communication of the ACM October 1999/Vol.42, No.10.

[10]. Mattsson, M. and Bosch, J. Observations on the Evolution of an Industrial OO Framework. Proceedings of ICSM'99, International Conference on Software Maintenance, Oxford, UK, 1999.

[11]. Microsoft COM Technologies. http://www.microsoft.com/com/.

[12]. MIME (Multipurpose Internet Mail Extensions) Part Three: Message Header Extensions for Non-ASCII Text. Request for Comments (RFC) 2047 http://www.rfc-editor.org/rfc/rfc2047.txt.

[13]. Pree, W. Design Patterns for Object-Oriented Software Development. Addison-Wesley Pub Co, March 1995.

[14]. Ripper, P., Fontoura, M. F., Neto, A. M., and Lucena, C. J. V-Market: A Framework for e-Commerce Agent Systems. World Wide Web, Baltzer Science Publishers, 3(1), 2000.

[15]. Foro oficial de Sun sobre todo tipo de dudas acerca de Java http://forum.java.sun.com/

[16]. Enciclopedia libre de internet http://es.wikipedia.org

[17]. Sun Microsystems, Inc Java 2 Enterprise Edition http://java.sun.com/j2ee/overview.html

[18]. Open Source Web Frameworks http://java-source.net/open-source/web-frameworks

[19]. Eric.j. Braude *Ingeniería del Software, una perspectiva orientada a objetos.* Madrid: RAMA, 2003 ISBN**:** 8478975756

[20]. Documentación asignatura *Ingeniería del Software I*. Área de conocimiento: Lenguajes y Sistemas Informáticos. Universidad Carlos III de Madrid.

[21]. Documentación asignatura Desarrollo de aplicaciones distribuidas, Tema Desarrollo de aplicaciones distribuidas en Java . Área de conocimiento: Arquitectura y Tecnología de Computadores. Universidad Carlos III de Madrid.

[22]. Patrones de diseño para el desarrollo de aplicaciones J2EE http://java.sun.com/blueprints/corej2eepatterns/ y http://www.corej2eepatterns.com

[23]. El modelo de DAOs de Puzzle se basa en el patrón DAO propuesto en los patrones de J2EE http://java.sun.com/blueprints/patterns/DAO.html

[24]. JOTM (Java Open Transaction Manager) se presenta como software libre que implementa un gestor transaccional que cumple las especificaciones JTA y protocolo XA: http://jotm.objectweb.org

[25]. Log4j Framework de software libre que proporciona un servicio de trazas. http://logging.apache.org/log4j/docs/manual.html

[26]. Wikipedia: La enciclopedia de contenido libre en Internet. http://es.wikipedia.org/wiki/Wikipedia:Portada